NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

EVOLVED DESIGN, INTEGRATION, AND TEST OF A MODULAR, MULTI-LINK, SPACECRAFT-BASED ROBOTIC MANIPULATOR

by

Jerry V. Drew II

June 2016

Thesis Advisor:	Marcello Romano
Second Reader:	Josep Virgili-Llop

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE EVOLVED DESIGN, INTEGRATION, AND TEST OF A MODULAR, MULTI-LINK, SPACECRAFT-BASED ROBOTIC MANIPULATOR	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jerry V. Drew II	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis reports on the evolved design, test, and integration of a robotic manipulator consisting of multiple modular links, which enable the reconfiguration of the manipulator system for differing mission requirements without constructing unique hardware for each experimental campaign.

The evolved design replaced custom components with commercial components to improve performance, standardize hardware, and reduce assembly time. Additional links were constructed and assembled into a four-link manipulator capable of moving its end-effector without imparting motion to the base spacecraft. Each joint can be controlled independently and provides unique telemetry data via Wi-Fi.

A mathematical model of the system was implemented, and the kinematic and dynamic behaviors calibrated, resulting in confirmation of the validity of the modular link manipulator concept. A software code based on this model, the Spacecraft Robotics Toolkit (SPART), was published as an open-source kinematics/dynamics and control framework for use by the spacecraft robotics community. Future research will investigate further upgrades, manipulator control and use in operational scenarios.

14. SUBJECT TERMS spacecraft, robotics, kinematics, dynamics, multi-body mechanics	**15. NUMBER OF PAGES** 177
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

EVOLVED DESIGN, INTEGRATION, AND TEST OF A MODULAR, MULTI-LINK, SPACECRAFT-BASED ROBOTIC MANIPULATOR

Jerry V. Drew II
Captain, United States Army
B.S., United States Military Academy, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2016

Approved by: Marcello Romano
Thesis Advisor

Josep Virgili-Llop
Second Reader

Garth Hobson
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis reports on the evolved design, test, and integration of a robotic manipulator consisting of multiple modular links, which enable the reconfiguration of the manipulator system for differing mission requirements without constructing unique hardware for each experimental campaign.

The evolved design replaced custom components with commercial components to improve performance, standardize hardware, and reduce assembly time. Additional links were constructed and assembled into a four-link manipulator capable of moving its end-effector without imparting motion to the base spacecraft. Each joint can be controlled independently and provides unique telemetry data via Wi-Fi.

A mathematical model of the system was implemented, and the kinematic and dynamic behaviors calibrated, resulting in confirmation of the validity of the modular link manipulator concept. A software code based on this model, the Spacecraft Robotics Toolkit (SPART), was published as an open-source kinematics/dynamics and control framework for use by the spacecraft robotics community. Future research will investigate further upgrades, manipulator control and use in operational scenarios.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
- **A. RESEARCH MOTIVATION 1**
- **B. STATE OF THE ART 2**
  - **1. An Overarching Context 2**
  - **2. Spacecraft Based Manipulators in Experiment and Application 5**
  - **3. Efforts at the Naval Postgraduate School (NPS) Spacecraft Robotics Laboratory (SRL) 10**
- **C. RESEARCH OBJECTIVES 13**

II. DESIGN EVOLUTION 15
- **A. REQUIREMENTS 15**
- **B. MODIFIED DESIGN 15**
  - **1. Retained Hardware 15**
  - **2. The Link Structure 16**
  - **3. Hardware Upgrades 27**

III. ASSEMBLY DOCUMENTATION 35
- **A. INTENT OF THE ASSEMBLY DOCUMENTATION SECTION 35**
- **B. BUILDING A LINK 36**
  - **1. The Servomotor Cable Connectors 36**
  - **2. The Encoder Cable Connector 39**
  - **3. The Torque Sensor Wiring 42**
  - **4. Battery Connector and DC/DC Converter 44**
  - **5. The Arduino Subassembly 44**
  - **6. Mounting the Components to the Structure 45**

IV. DATA ARCHITECTURE 59
- **A. OVERVIEW 59**
- **B. CONCEPT OF OPERATIONS 59**
- **C. DATA FLOW 60**
  - **1. The D-Link Interface 60**
  - **2. The Driver Interface and the Arduino Due Code 62**
  - **3. The Simulink Model 64**
  - **4. The VICON Interface 68**
  - **5. Potential for an *Ad Hoc* Network 70**

V. KINEMATICS AND DYNAMICS CALIBRATION 71
A. GENERAL OVERVIEW 71
B. KINEMATICS OVERVIEW 71
1. Definition of Terms 71
2. The Base 72
3. The Joints 73
4. Rotation and Joint Transformation Matrices 73
5. Denavit-Hartenberg (DH) Parameters 75
6. Link Transformation Matrices 77
C. DYNAMICS OVERVIEW 77
1. Twist Propagation Matrices (B_{ij} and B_{i0}) 78
2. Velocity Transformation Matrix (P_0) and Twist Propagation Vector (p_m) 78
D. KINEMATIC CALIBRATION 80
E. DYNAMIC CALIBRATION 81
1. Overview 81
2. Inertia Matrices 82
3. Mass Matrices 83
4. Generalized Inertial Matrices 83
5. Determining the Base-Manipulator System State (q) 84
6. Inducing Variation, the Dynamic Calibration Proper 85
F. ANALYSIS 85

VI. CONCLUSION 89
A. SUMMARY OF WORK 89
B. FUTURE WORK 89
C. RESEARCH SIGNIFICANCE 93

APPENDIX A. SPARKFUN HX711 LOAD CELL AMPLIFIER OPEN-SOURCE HEADER FILE [51]..........95

APPENDIX B. SPARKFUN HX711 LOAD CELL AMPLIFIER OPEN-SOURCE C++ SOURCE CODE FILE [51]..........97

APPENDIX C. ONBOARDLINK.INO..........101

APPENDIX D. MARSMAN_DRIVER_LOAD.CCX..........111

APPENDIX E. VICON_CALIBRATE.M..........115

APPENDIX F. KINEMATICS_SERIAL.M..........119

APPENDIX G. DH_RS.M..........123

APPENDIX H. SKEWSYM.M..........125

APPENDIX I. MARSMAN_KINCAL.M..........127

APPENDIX J. X_EE.M..........133

APPENDIX K. MARSMAN_DYNCAL.M..........135

APPENDIX L. I_I.M..........139

APPENDIX M. MCB_SERIAL.M..........141

APPENDIX N. GIM_SERIAL.M..........143

APPENDIX O. Q_DOT_FUN.M..........145

APPENDIX P. Q0_MANEUVER.M..........147

LIST OF REFERENCES..........149

INITIAL DISTRIBUTION LIST..........155

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Two Baxter Robots with Different End Effectors. Source: [7]. 4

Figure 2. Precise Surgical Manipulators. Source: [9]. 5

Figure 3. Deployment of the original Canadarm from *Columbia*. Source: [11]. 6

Figure 4. EFFORTS-1 Floating Via Air Bearings. Source: [13]................................7

Figure 5. Astronaut Reference Flying Robot (ARFR). Source: [15]. 8

Figure 6. ETS-VII. Source: [17]...9

Figure 7. DYMAFLEX with Deployed Antennas. Source: [19]..............................10

Figure 8. The First SRL Multi-link Manipulator. Source: [20]................................11

Figure 9. Complete Link Assembly. Source: NPS SRL...12

Figure 10. The Floating Spacecraft Simulator (Without Reaction Wheel). Source: NPS SRL...13

Figure 11. Five Pieces of the Main Link Structure with Substrate. 17

Figure 12. Five Pieces of the Main Link Structure without Substrate. 17

Figure 13. FHA-8C-30-12S17b-E Servomotor. 18

Figure 14. DEP-090-09 Servomotor Driver. Source: [25]. 19

Figure 15. FUTEK TFF400 Torque Sensor, Isometric and Rear Views. Source: [26]..20

Figure 16. Wiring Diagram for FUTEK's TFF400 Torque Sensor. Source: [30]. 21

Figure 17. The Inspired Energy NH2054 Li-Ion Battery. Source: [27]. 22

Figure 18. Discharge Curve of 14.4V Li-Ion Battery. Source: [27] 23

Figure 19. Traco Power 75-2415WI DC/DC Converter. Source: [28]. 24

Figure 20. Arduino Due Microcontroller. Source: [31]. 25

Figure 21. LinkSprite RS232 Shield. Source: [32]. 25

Figure 22. Dantona 3.6V Battery. 26

Figure 23. Air Delivery System Components. 27

Figure 24. Both sides of the SparkFun Load Cell Amplifier. Source: [33] 28

Figure 25. Arduino Wi-Fi Shield. Source: [35]...31

Figure 26. Voltage Regulator. Adapted from [36]. 33

Figure 27. "Shoe" Developed Using the Free Microsoft 3D Builder Application......34

Figure 28. Servomotor Cabling Schematic. Source: [25]. 36

Figure 29. Servomotor with Stock Connectors. 37

Figure 30. Motor Lead Connector Layout (Units in mm). Source: [39]. 38

Figure 31. Motor Cable Wiring with 4-Position J2 Connector. 39

Figure 32. Pinout of Encoder Cable with Stock Connector. Source: [39]. 40

Figure 33. Replacement Connector Pieces Alongside the Encoder Cable. 41

Figure 34. Pinout of Servomotor Encoder Cable with 20-Pin Connector. 42

Figure 35. Encoder Cable with 20-Pin Connector and Attached Battery. 42

Figure 36. Torque Sensor Wired to the Load Cell Amplifier. 43

Figure 37. Block Diagram of the Electrical Power Architecture. 44

Figure 38. Torque Sensor Mounted on the Motor Carriage. 46

Figure 39. Servomotor Mounted to Outer Plate 47

Figure 40. Mated Torque Sensor and Servomotor 47

Figure 41. Circular Bearings for Use on Outer Plate and Outer Bracket. 48

Figure 42. The Completed Joint Assembly. 48

Figure 43. Diagram of the Harmonic Drive Connection Ports. Source: [41]. 50

Figure 44. Link with Joint Assembly, Mounted Servomotor Driver and DC/DC Converter...51

Figure 45. Copley Motion Explorer 2 (CME2) Graphic User Interface upon Initialization. 52

Figure 46. CME2 Control Panel with Disabled Motor Control. 53
Figure 47. CME2 Control Panel, Faults Cleared, Jog Enabled. 54
Figure 48. Power Architecture Schematic 55
Figure 49. Data Architecture Schematic 56
Figure 50. Complete Link Assembly. Source: NPS SRL 57
Figure 51. Diagram of the Experimental Setup. Adapted from [42]. 60
Figure 52. Code Defining Network Parameters, Port Numbers, and MAC Addresses. 62
Figure 53. Send/Receive Switch for Link Control. 64
Figure 54. Breakout of the Data Processing Subsystem. 65
Figure 55. Breakout of the Checksum Subsystem. 66
Figure 56. Breakout of the Enabled Subsystem. 66
Figure 57. Capture Function 67
Figure 58. Experimental Setup Including VICON Cameras. Source: [44]. 68
Figure 59. Silver Spheres Serve as Markers for VICON. Source: [44]. 69
Figure 60. Screenshot of the VICON Tracking Software. 70
Figure 61. Visual Depiction of the Denavit-Hartenberg Parameters. Source: [44]. 75
Figure 62. Depiction of Dynamic Model Vectors: Source: [48]. 79
Figure 63. Dynamic Calibration Matrix. 85
Figure 64. MATLAB Results of the Kinematics Calibration. 86
Figure 65. MATLAB Outputs of the Dynamic Calibration. 87
Figure 66. Sample End Effector from the Yale OpenHand Project. Source: [50]. 91

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Servomotor and Driver Characteristics. Sources: [20], [25], [26]. 19

Table 2. Torque Sensor Characteristics. Sources: [4], [20]. 21

Table 3. ASCII Strings for Due-to-Driver Communication. Source: [25]..............63

Table 4. Denavit-Hartenberg Parameters. Adapted from [44]. 75

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

A	ampere
AAS	American Astronautical Society
ADC	analog-to-digital converter
Ah	ampere-hours
AIAA	American Institute for Aeronautics and Astronautics
ARFR	Astronaut Reference Flying Robot
ASCII	American Standard Code for Information Interchange
CAD	computer-aided drafting
ccp	complimentary C++
ccx	Copley Controls Axis
CLK	clock
CME	Copley Motion Explorer
COTS	commercial-off-the-shelf
DAE	differential algebraic equation
DARPA	Defense Advanced Research Projects Agency
DAT	data
DC	Direct Current
DH	Denavit-Hartenberg
DHCP	dynamic host configuration protocol
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DOF	degrees of freedom
DYMAFLEX	Dynamic Manipulation Flight Experiment Microsat
EFFORTS	Experimental Free-Floating Robot Satellite
EtherCAT	Ethernet for control automation technology
ETS-VII	Experimental Test Satellite 7
EXEC	excitation
FSS	Floating Spacecraft Simulator
GNC	guidance, navigation, and control
GUI	graphic user interface
ICATT	International Conference on Astrodynamics Tools and Techniques

ICRA	International Conference on Robotics and Automation
IEEE	Institute of Electrical and Electronics Engineers
IP	internet protocol
IROS	Intelligent Robots and Systems
Li-ion	Lithium-ion
LDO	low dropout regulator
MAC	media access control
MARSMAN	Modular Arm Robotic Manipulator
MATLAB	Matrix Laboratory
MIT	Massachusetts Institute of Technology
NPS	Naval Postgraduate School
OEDMS	Orbital Express Demonstration Manipulator System
PE	protected earth
R	resistance
ROTEX	Robot Technology Experiment
RS232	Recommended Standard 232
Rx	receive
SIG	signal
SPART	Spacecraft Robotics Toolkit
SRL	Spacecraft Robotics Laboratory
SRMS	Shuttle Remote Manipulator System
STS	Space Transportation System
Tx	transmit
TTL	transistor-transistor logic
UART	universal asynchronous receiver/transmitter
UAV	unmanned aerial vehicle
UDP	Unit Datagram Protocol
USC	University of Southern California
V	volt

ACKNOWLEDGMENTS

There are necessarily many people to thank when one completes a single project that spans an entire year and multiple areas of expertise.

First, to Dr. Marcello Romano, my advisor: Thank you for your oversight and guidance throughout the process. I appreciate all the work you have done over the years to assemble a state-of-the-art facility and to fill it with high-quality people.

Second, to my second reader, Dr. Josep Virgili-Llop, who taught me something new almost every day: I appreciate your patience and your willingness to drag me along the many different lines of investigation.

Third, I would like to thank the members of the Spacecraft Robotics Lab and my classmates in the Astronautical Engineering curriculum, whose help and support over the last 21 months has been invaluable.

Most importantly, I need to thank my wife, Meagan, and our wonderful children—Jerry, Clara, Marian, and Laurel—for their unwavering support. All of this would not be worth it without you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. RESEARCH MOTIVATION

The uses of robotics in the modern world are nearly limitless. From manufacturing applications to high-precision surgery, from unmanned aerial vehicles (UAVs) to planetary rovers to satellites with manipulator arms, applications for robotics abound. Within any one of these application areas, specialized areas of investigation emerge. A roboticist must strive to understand hardware like motors and sensors, software for control and simulation, the under-lying multi-body mechanics, and the various techniques for implementing these pieces into a functional system (mechatronics). In areas that are inherently dangerous or inaccessible to humans, like outer space, robotic applications are a natural fit.

Since the 1980s, the notion of using robotic servicing satellites to extend the lifetime of on-orbit assets has generated an increasingly large field of interest. Traditional satellites continue to be very large and very expensive, typically the result of many years of substantial investment. During a satellite's operational lifetime, any number of electrical, mechanical, thermal, propulsion or structural problems may arise and render it incapable of performing its mission. A servicing satellite equipped with a robotic manipulator may be able to correct or mitigate these problems, and this technology has been demonstrated, most notably in Japan's Experimental Test Satellite 7 (ETS-7), and the United States' Orbital Express mission. Servicing satellites such as these could assist in the deployment of a stuck antenna or solar panel, the replenishment of an empty propellant tank, or the stabilization of a tumbling spacecraft [1], [2]. For failed spacecraft and for other pieces of space junk, a satellite with a manipulator arm could gain control of the debris and assist its deorbit or tug it to a location where it will not interfere with active missions.

Among terrestrial manipulators, fixed-based manipulators for use in industrial applications are the most common type. Over the years, the mechanics and control theory necessary to operate these industrial systems were adapted for use with spacecraft manipulators, taking into account the very important consideration that a satellite with a manipulator cannot be treated as a fixed-base manipulator in all cases. If, for example, the manipulator is sufficiently large with respect to the base spacecraft, or if the manipulator moves faster than a given speed, the action of the arm will impart significant reaction onto the base and vice versa.

To account for this highly nonlinear dynamic coupling, then, the kinematics and the dynamics of the system must be modeled and simulated numerically to understand how the system will behave for a given number of manipulator links. Further, since one cannot simulate all physical effects of the environment (friction forces or sensor noise, for example), a hardware validation of the model is necessary. Because of the high mass penalty of putting spacecraft into orbit, it is desirable to launch the smallest satellite capable of performing a given mission—even if using a smaller satellite means greater complexity in the dynamical behavior of the system. With this goal in mind, research at the Naval Postgraduate School (NPS) Spacecraft Robotics Laboratory (SRL) on free-floating spacecraft simulators with multiple-link robotic arms has been ongoing.

B. STATE OF THE ART

1. An Overarching Context

Space-based manipulators and the experimental models used for development and simulation have a long heritage, but before discussing the very specific application of a satellite-based robotic manipulator, a discussion of the definition of robotics and consideration of the state-of-the art of robotics is useful. The experimental and operational heritage of the current research effort is discussed, including the early examples of floating manipulators, manipulator campaigns aboard the Space Transportation System (STS) and the International Space Station (ISS), the Experimental Test Satellite 7 (ETS-VII) and Orbital Express missions, and ongoing research efforts within academia.

In her book, *The Robotics Primer,* Dr. Maja Matarić, the Vice Dean for Research at the University of Southern California's (USC) Viterbi School of Engineering, rigidly defines a robot as "an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals" [3]. While this definition allows for systems of varying levels of autonomy, Matarić goes on to deliberately exclude manipulators that are controlled via tele-operation—that is, an operator controls the manipulator remotely. This exclusion contradicts a reality that is evident in published literature: the larger robotics community considers tele-operated manipulators as robots. As evidence, the 2015 Institute of Electrical and Electronics Engineers' (IEEE) International Conference on Robotics and Automation (ICRA) and International Workshop on Intelligent Robots and Systems (IROS), two very prominent conferences, included multiple papers on tele-operation, particularly as it relates to high-precision surgery (for example, [4], [5]) but also relating to space operations [6]. Indeed, as a general observation, the literature of space manipulators (tele-operated or not) proclaims their inclusion in the field of robotics. Therefore, this thesis keeps with the larger community in its consideration of tele-operated manipulators as a valid subset of robotics.

Even highly independent, non-tele-operated robots depend upon a human user for varying degrees of input (that is, they have varying levels of autonomy). At the more sophisticated level, Rethink Robotics touts its industrial robot Baxter as "trained, not programmed" [7]. Baxter is a robot that specializes in pick-and-place tasks, but unlike other robots, a user can reconfigure Baxter to perform a new task simply by manually repositioning its manipulators in the desired sequence [8]. If an object is out of its expected place—as it moves along a conveyor belt, for example—a camera mounted in Baxter's wrist will seek the misplaced object and automatically adjust its behavior, and if a human co-worker gets in the way, Baxter will sense the collision and safely cease operation [8].

Two versions of the Baxter robot with different end effectors are shown in their industrial setting in Figure 1. Industrial robots offer two sharp contrasts to the robotics found on spacecraft. First, Baxter has a fixed base, so motion of its manipulators does not impart a reaction motion by the base. Second, the complexity of Baxter's behavior is not an option when dealing with a manipulator on an unmanned satellite because there is no human nearby to "teach" it. Since autonomous operation of a satellite-based robotic manipulator is a highly sophisticated endeavor, most spacecraft robotic manipulators have historically been operated via tele-operation.

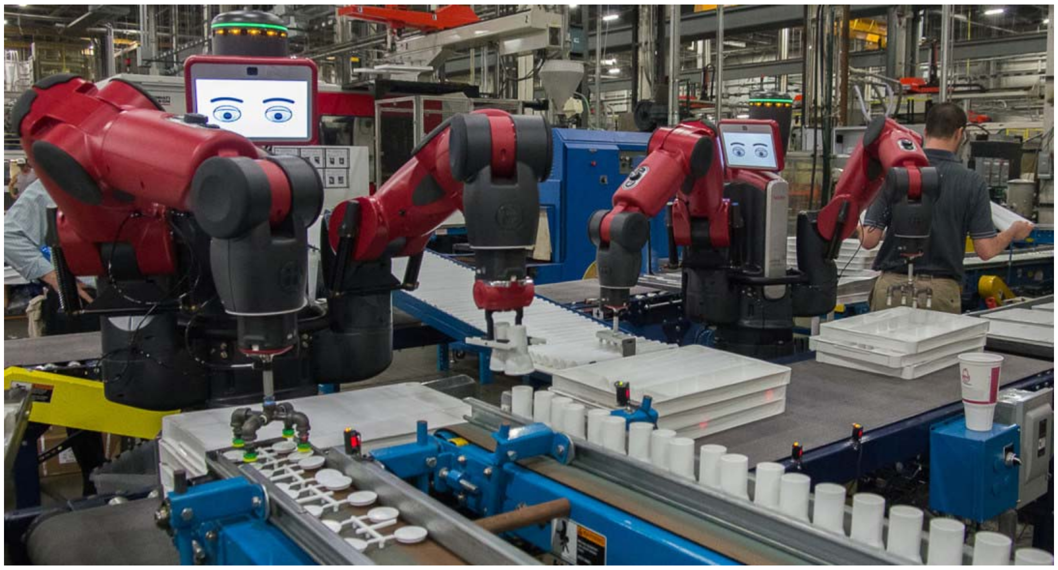


Figure 1. Two Baxter Robots with Different End Effectors. Source: [7].

Earth-bound tele-operated robots, like Baxter and other more autonomous earth-bound manipulators, are (not surprisingly) capable of higher precision and higher complexity functions than their space-bound counterparts. In particular, the field of surgery via tele-robotics has flourished, but efforts are underway to make the process more autonomous. The manipulators pictured in Figure 2 employ as end effectors a 6 mm grasper and a 10 mm pair of scissors to remove simulated tumors from simulated tissue (left) and to remove simulated damaged skin (right) [9]. These manipulators are programmed by a technique that “involves observing human-operated demonstrations into motion sequences and transition conditions” and are capable of detecting the simulated tumors and dead skin and acting accordingly [9]. Like the manual repositioning of Baxter, learning by observation presents intriguing possibilities for the field of spacecraft robotics, but because of the dynamic behavior of a spacecraft-manipulator system, such a system will likely never be able to achieve the precision of surgical robotics.

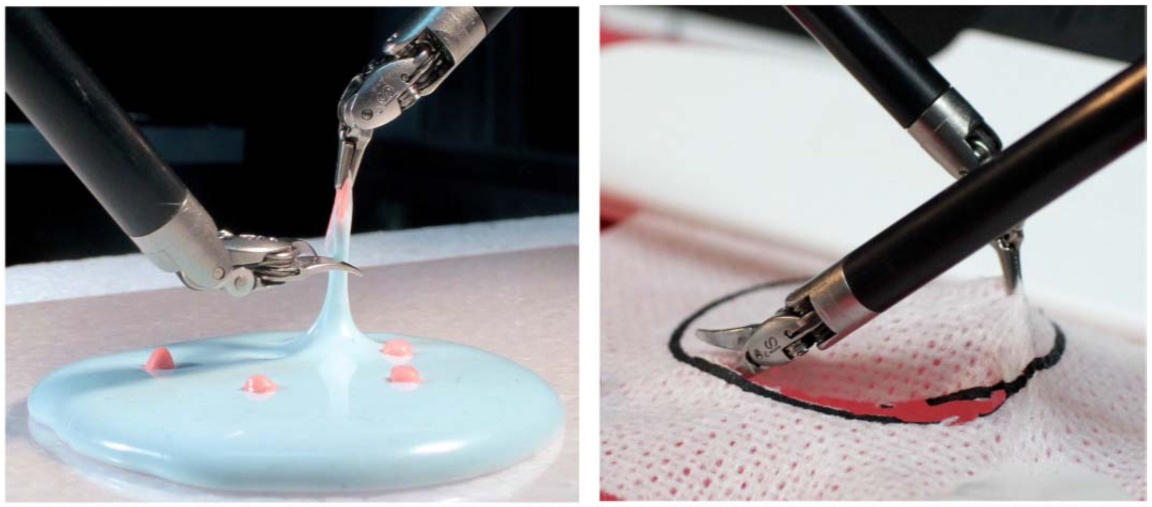


Figure 2. Precise Surgical Manipulators. Source: [9].

2. Spacecraft Based Manipulators in Experiment and Application

With examples of highly-programmable and highly precise robotic manipulators in mind, it is now useful to consider a brief history of spacecraft-based manipulators, both those used in the simulation environment and those actually employed in space. The most recognized space manipulator is the Shuttle Remote Manipulator System (SRMS) or the Canadarm, the original version of which flew on the second Shuttle mission (STS-2, *Columbia*) in 1981 (Figure 3) [10]. The four subsequent Canadarms, including the current version aboard the International Space Station (colloquially referred to as Canadarm2), have employed programmed routines and tele-operation by astronauts [11]. According to interviews with former astronauts Dr. Jim Newman and Captain (Retired) Dan Bursch, the light weight of the manipulators relative to their host spacecraft, their relatively slow motion, and carefully planned maneuvers mitigate the problem of dynamically coupling the manipulator's motion to that of the spacecraft.

Because of its enormous size, the ISS is less susceptible to dynamic coupling from its manipulators than was the Shuttle. Specifically, "Canadarm2 has a mass of 1,800 kg, in comparison to the ISS, which has a mass of approximately 420,000 kg," and its tip speed "during station assembly is 2 cm/s, while during [Extra Vehicular Activity] support the maximum speed is 15 cm/s" [12]. In essence, the Canadarm2 is so small relative to the ISS and moves so slowly within its preplanned maneuver envelope that the system can be treated as a fixed-based manipulator from the perspective of system dynamics. However, with the trend toward smaller, more capable satellites in mind, questions about how a manipulator's motion would affect the dynamics of a "small" spacecraft became an area ripe for investigation.

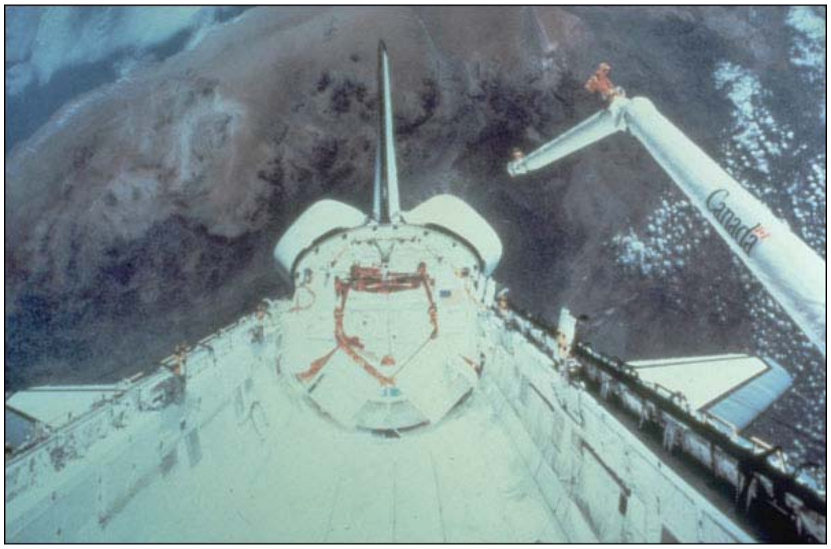


Figure 3. Deployment of the original Canadarm from *Columbia*. Source: [11].

One of the earliest experimental campaigns on small spacecraft-manipulator systems began in 1987 under Dr. Kazuya Yoshida at Japan's Tohuku University with the Experimental Free-Floating Robot Satellite (EFFORTS-1, Figure 4) [13]. By this time, experimentation on spacecraft robotics had taken place in multiple different experimental environments including parabolic airplane flight, neutral buoyancy pools, and tethered suspension [13]. The best testing environment, of course, would have been outer space, but limited access prevented such an opportunity to all but a few. The most practical method—the one employed by Yoshida, by the experimental campaign for this thesis, and still the most common method for spacecraft robotics experimentation [14]—was employing a flat, smooth surface over which a robot could float on air bearings.

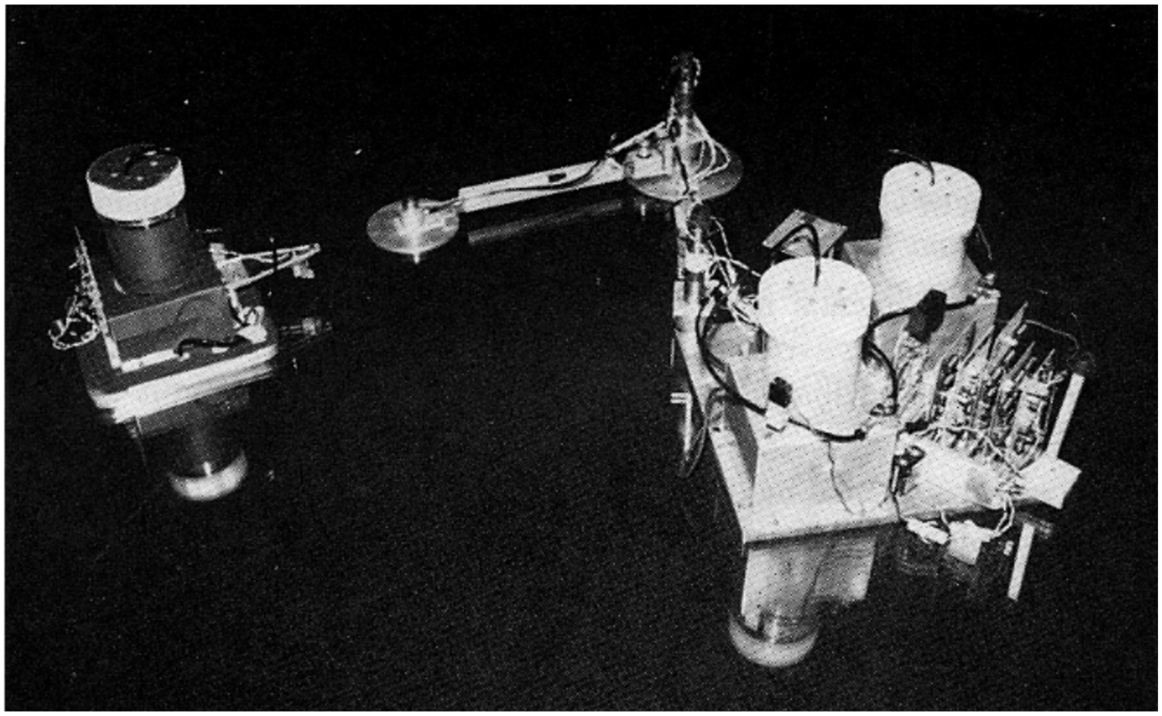


Figure 4. EFFORTS-1 Floating Via Air Bearings. Source: [13].

In 1992, the Japanese research team of Kazuo Machida, Yoshitsugu Toda, and Toshiaki Itawa published results of their experimentation on a multiple-armed spacecraft simulator, the Astronaut Reference Flying Robot (ARFR, Figure 5) [15]. It was a tele-operated system aimed at "performing in-orbit servicing in the manner of an astronaut with a manned maneuvering unit" [15]. Like EFFORTS-1, ARFR was developed as a free-floating robot. Meanwhile, Yoshida was working on EFFORTS-2, a two-armed floating spacecraft-manipulator system that was functionally similar to the ARFR, including its dependence on tele-operation [13]. Experimental campaigns resembling ARFR and EFFORTS would continue throughout the 1990s—many at the university level.

One experimental campaign that was fortunate enough to be tested in space was Germany's Robot Technology Experiment (ROTEX). In 1993, ROTEX flew aboard the Shuttle and demonstrated the capability for automatic behavior, tele-operation from the Shuttle, tele-operation from Earth and "tele-sensor programming (i.e., learning by showing in a completely simulated world on-ground, including the sensory perception with sensor-based execution later on-board)" [16]. As a small, on-board manipulator, the dynamics of ROTEX, like the dynamics of the unloaded Canadarm, were inconsequential to the dynamics of the spacecraft, but the ROTEX experiment demonstrated that automation of robotics, the tele-operation of robotics, and the desire to combine these technologies on a small spacecraft were developing in parallel. In fact, these technologies were actually converging.

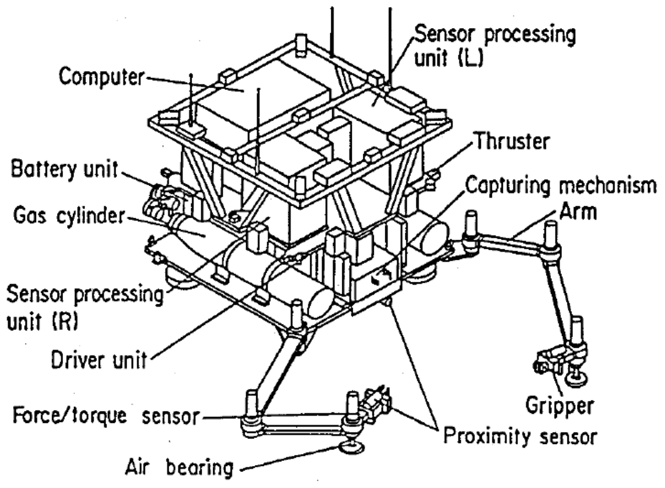


Figure 5. Astronaut Reference Flying Robot (ARFR). Source: [15].

The initial convergence came in the form of Japan's Engineering Test Satellite 7 (ETS-VII), which was launched November 28, 1997 (Figure 6) [17]. ETS-VII actually consisted of two satellites, a 2.5-ton chaser and a 0.4-ton target spacecraft [1] and accomplished the "autonomous [rendezvous and docking] by [an] unmanned space vehicle for [the] first time in the world" [18]. With a relative mass between the spacecraft and the manipulator of 24, the manipulator was small enough that its motion did not significantly affect the motion of the base [12]. Further, the robotic manipulator missions were conducted by tele-operation from the ground [17]. Thus, while ETS-VII was a very important step in the evolution, it falls short of a small spacecraft with relatively large manipulator capable of autonomous behavior.

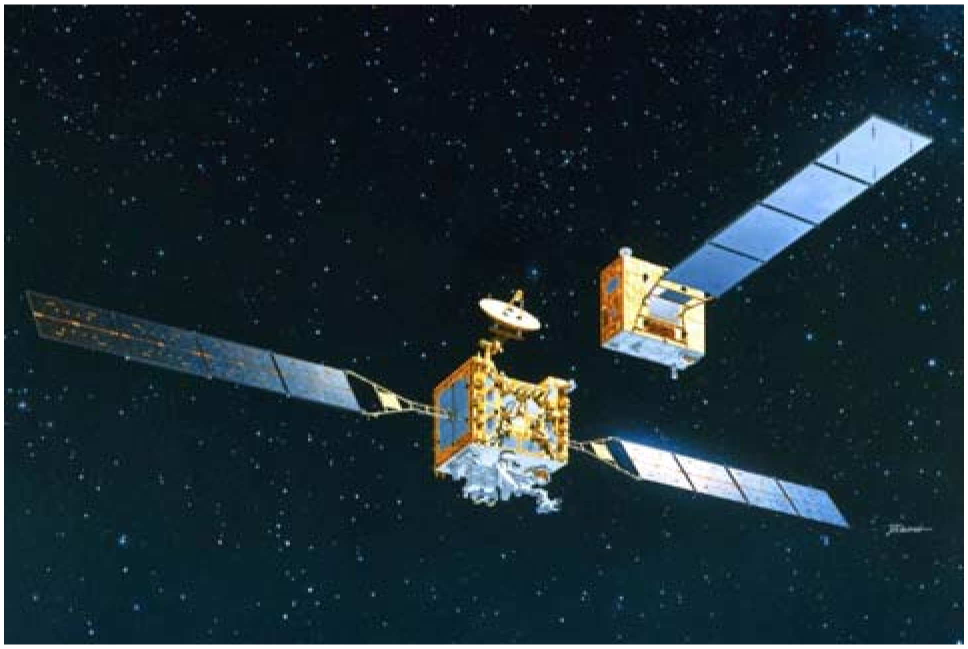


Figure 6. ETS-VII. Source: [17].

In March of 2007, the Defense Advanced Research Project Agency (DARPA) of the United States launched the Orbital Express mission [2]. In many ways, the experimental campaign resembled that of ETS-VII. Orbital Express consisted of a chaser vehicle named ASTRO with a six-degree of freedom manipulator named the Orbital Express Demonstration Manipulator System (OEDMS) and a target vehicle named NextSat [2]. Like ETS-VII, the relatively large size of ASTRO compared to OEDMS—a relative mass ratio of 15—prevented significant effects of the manipulator's motion from affecting the base spacecraft itself [12], Unlike ETS-VII, however, Orbital Express was capable of conducting operations with "various levels of autonomy," including "operations where only a single command was sent to initiate the test scenario" [2]. Space-based robotic manipulators without tele-operation had arrived.

A spacecraft with a small relative mass compared to its manipulator has yet to fly in space, but efforts are underway at the University of Maryland's Space Systems Laboratory to develop the Dynamic Manipulation Flight Experiment Microsat (DYMAFLEX, Figure 7). Unlike ETS-VII or Orbital Express, DYMAFLEX is a small satellite with a robotic manipulator that is ~14% of the mass of the combined system [19] (a relative mass ratio of 5.3) [11]. Such a small satellite would provide a cheaper alternative to launching behemoth satellites with relatively small manipulators, and if the coupled dynamics can be controlled and the autonomous behavior accomplished, DYMAFLEX could represent the convergence of small satellite and autonomous robotics sought since Yoshida began his experiments in the 1980s.

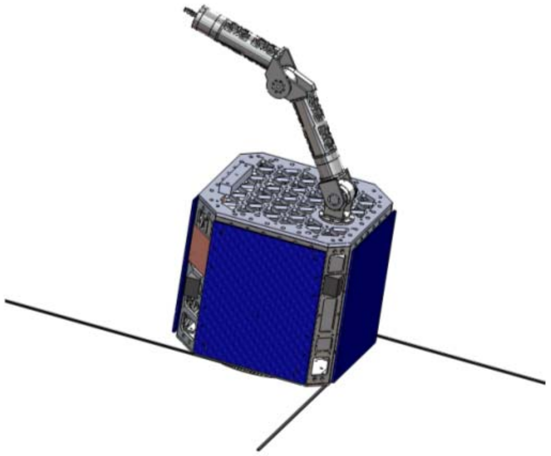


Figure 7. DYMAFLEX with Deployed Antennas. Source: [19].

3. Efforts at the Naval Postgraduate School (NPS) Spacecraft Robotics Laboratory (SRL)

The current efforts at the NPS SRL follows the same basic approach as many other robotics efforts throughout academia. The basic methodology consists of four phases: (1) design and build a manipulator, (2) attach it to a base spacecraft simulator, (3) characterize its kinematic and dynamic behavior, and (4) control the manipulator to perform a value-added task. The first SRL multi-link manipulator program was a four-link manipulator constructed in 2012 (Figure 8).

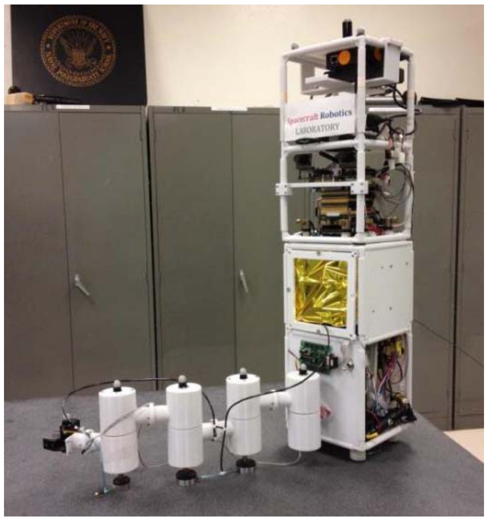


Figure 8. The First SRL Multi-link Manipulator. Source: [20].

In this same year, the 16 m^2 polished granite monolith was installed in the laboratory, providing additional maneuver space over the legacy 9 m^2 monolith. The table-top method experimentation allows for a spacecraft/manipulator system that operates in three degrees of freedom (translation in the *x* and *y* directions of the Cartesian plane and rotation about the *z* axis, which is perpendicular to the table. A four-joint robotic manipulator system is desirable because it offers four degrees of freedom (4DOF). Despite the dynamic coupling, a 4DOF manipulator operating in a 3DOF experimental environment enables the base to remain stationary under the right circumstances. Such kinematic redundancy is convenient if, for example, the host spacecraft has a camera that needs to observe the end effector working.

The first-generation manipulator system was designed to attach to the lab's third generator Floating Spacecraft Simulator (FSS), the tall rectangular prism in Figure 8. As in nearly all manipulators, the links depended upon the base spacecraft for power and computation, requiring electrical wiring to be strung throughout the link. Once constructed, the manipulator could not be reconfigured, limiting the number and type of table-top experimental problems to which the system could be applied.

The solution to this limitation was to design and build a modular link that contained all of the power, hardware, and software within its own structure (Figure 9).

These independent links are compatible with the new fourth-generation FSS (Figure 10), which was designed to allow manipulator mountings on each of its four faces. In the current configuration, the links depend upon the FSS only for the compressed air that flows through a series of detachable air hoses to the link air pads.

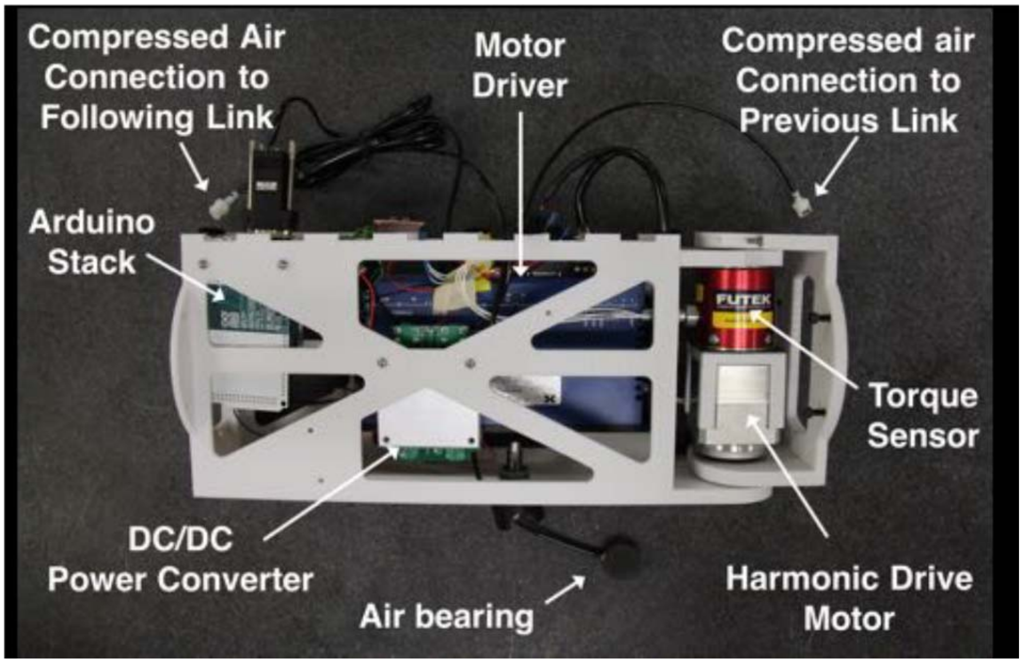


Figure 9. Complete Link Assembly. Source: NPS SRL.

Four modular, independent links can be reconfigured and mounted on the FSS in any number of ways. For example, all four links could be attached into one serial manipulator as in the current investigation, but two manipulators could just as easily be affixed to two adjacent (or two opposite) sides for experimentation with two cooperative (or two independent) manipulators such as EFFORTS-2, ARFR, the Massachusetts Institute of Technology's (MIT) Experimental Free-Flying Robot [21], and the Tsinghua University's Humanoid Free-flying Space Robot [22]. The two-armed manipulators provide examples of systems designed to have cooperative manipulators, but like their single manipulator counterparts, they lack the ability to be reconfigured. The cooperative control of both manipulators presents a challenge beyond the scope of this thesis, but is recommended as a future area of investigation.

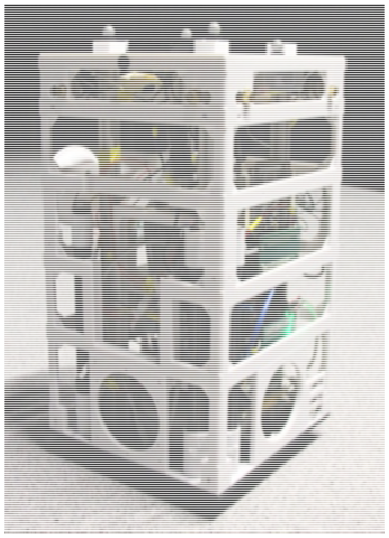


Figure 10. The Floating Spacecraft Simulator (Without Reaction Wheel). Source: NPS SRL.

The kinematics and dynamics of each configuration and the on-board control algorithms would change for different scenarios, but the hardware and the software to calibrate the kinematics and dynamics would remain largely the same. An entirely new manipulator would not need to be built for each series of experiments, which opens the possibility of multiple investigations without the time-consuming and expensive process of designing, building, testing, and configuring a new robotic manipulator every time the mission set changes. To that end, the first link of the SRL's second-generation spacecraft manipulator was designed and built by Dr. Markus Wilde and Mr. Daniel Alvarez from March 2013 to October 2014 [20]. At that time, the modular manipulator design (named the Modular Arm Robotic Manipulator, or MARSMAN) was believed by the researchers to be the first of its kind in the field. Dr. Josep Virgili-Llop, a National Research Council post-doctoral fellow at the NPS NRL, and the author continued work on the project beginning in June 2014.

C. RESEARCH OBJECTIVES

The primary purpose of the investigation was to achieve table-top experimentation of a workable, modular, four-link, single manipulator system and to calibrate the systems kinematics and dynamics. Although the temptation for continued design improvement was ever present, investigations into modified structural concepts was limited. Several significant hardware modifications were made, however, and these are explained in Chapter II. In the process, a 6DOF kinematics and dynamics calibration model was developed and validated (in 3DOF) through experimentation.

The 4-link MARSMAN configuration is believed to be the first multi-link manipulator composed of modular, reconfigurable links. It is also significant to note that the mass ratio of the base to the manipulator (approximately 1.3), and the inertia ratio of base to the manipulator (approximately 0.02 when the manipulator is fully extended) are significantly smaller than any of the systems previously discussed. These small numbers are indicative of a system with significant dynamic coupling and highly non-linear dynamics.

An open-source software model named the Spacecraft Robotics Toolkit (SPART) was presented at the 6th International Conference on Astrodynamics Tools and Techniques (ICATT) [23], and in an effort to reduce the duplication of efforts among roboticists worldwide, the software was made publicly available via the GitHub open-source, collaborative development environment [24]. To the knowledge of the author, SPART is the first open-source kinematics and dynamics simulator for spacecraft robotics.

With the four-link manipulator complete, the investigation is ready to proceed in any number of directions, including the control of the robotic manipulator, the integration of an end effector, or the use of the manipulator in capture and berthing scenarios. A more detailed discussion on the course of future work is offered in Chapter VI: Conclusion.

II. DESIGN EVOLUTION

This chapter recounts the evolution of the manipulator link design, highlighting the hardware and software changes between the first and subsequent manipulator links. While Link 1 was functional, there were multiple areas ripe for improvement. These design upgrades will be discussed in further detail, but prior to experimentation, Link 1 was upgraded with the same components as Links 2–4. Thus, all four links follow the same design.

A. REQUIREMENTS

The original design requirements were found to adequately meet the experimental goals of the project but were modified to employ Wi-Fi and commercial-off-the-shelf (COTS) components. The design requirements thus remain the same as those outlined by Wilde and Alvarez [20] with one modification and one addition:

- Easily interchangeable modular design
- No wires routed through joints
- **MODIFIED:** Wireless data relay to base robotic vehicle via Wi-Fi
- On-board power supply
- On-board servomotor and encoder
- On-board torque sensor
- Highest accuracy components for reasonable cost and mass
- **ADDED:** Use all commercial-off-the-shelf (COTS) components

B. MODIFIED DESIGN

1. Retained Hardware

The majority of the hardware components of the Link 1 were retained for subsequent links. These components include the 3D-printed link structure, the Harmonic Drive FHA-8C-30-12S17b-E servomotor and the Harmonic Drive DEP-090-09 servomotor driver [25], the FUTEK TFF400 torque sensor [26], the Inspired Energy NH254 lithium-ion battery [27], the Traco Power 75-2415WI V DC/DC converter [28].

2. The Link Structure

The structure of each manipulator link consists of five separate polycarbonate pieces designed in Siemens NX computer-aided design/computer-aided manufacturing/computer-aided engineering (CAD/CAM/CAE) software. Each piece was created in a part file format (.prt) and exported to the printer as a stereo lithography file (.stl).

Lesson Learned

While the NX software is very sophisticated, a freely available software package like the Windows 3D Builder application can be used to view and manipulate the .stl files. In this project, 3D Builder was used to create the link shoes (see Section II.3.e.). The default file format for items saved from 3D builder is the .3mf format, but objects can be saved as .stl files.

The five pieces of the link are the main link structure, the outer plate, the motor carriage, the inner bracket, and the outer bracket. Figure 11 shows the pieces as they arrive from the 3D printer. Figure 12 shows the same pieces after the substrate has been removed.

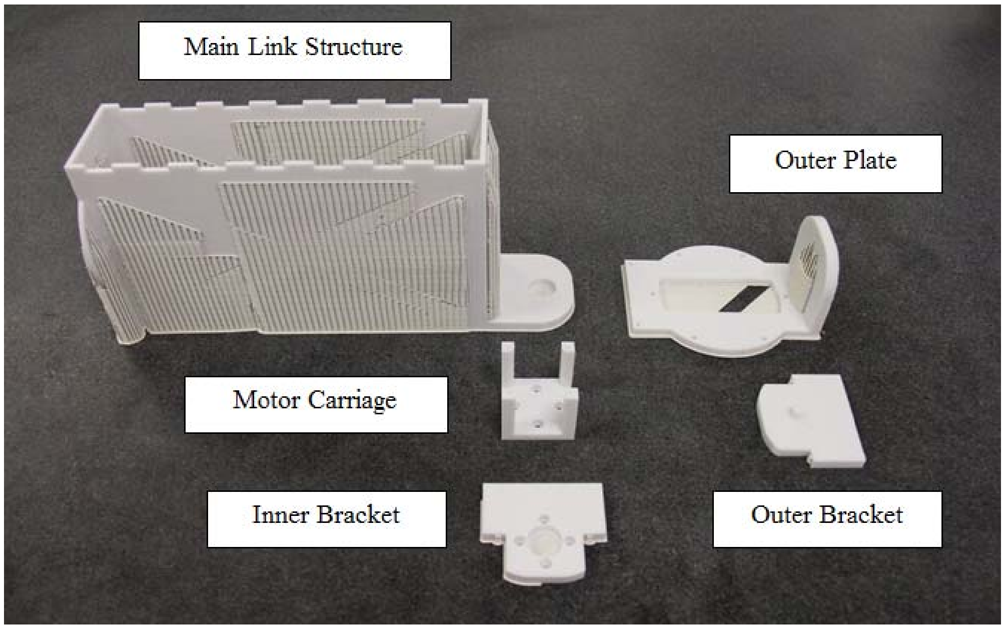


Figure 11. Five Pieces of the Main Link Structure with Substrate.

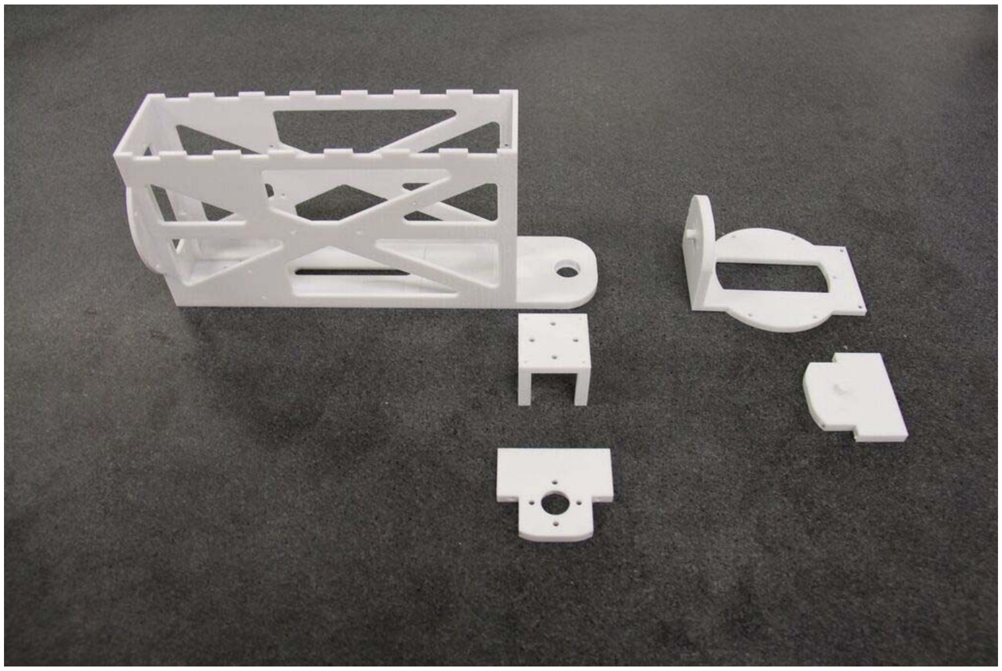


Figure 12. Five Pieces of the Main Link Structure without Substrate.

a. *The FHA-8C-30-12S17b-E Servomotor and the DEP-090-09 Servomotor Driver*

The FHA-8C-30-12S17b-E harmonic drive servomotor (Figure 13) is the on-board actuator for each manipulator link. It responds to commands from the DEP-090-09 servomotor driver (Figure 14)—a driver chosen specifically for its compatibility with the servomotor's absolute encoder. The servomotor sends its position data back through the servomotor driver. The driver calculates velocity based upon the position data and notes the amount of applied current (a proxy for applied torque). The controller uses these kinematic data points for each link.

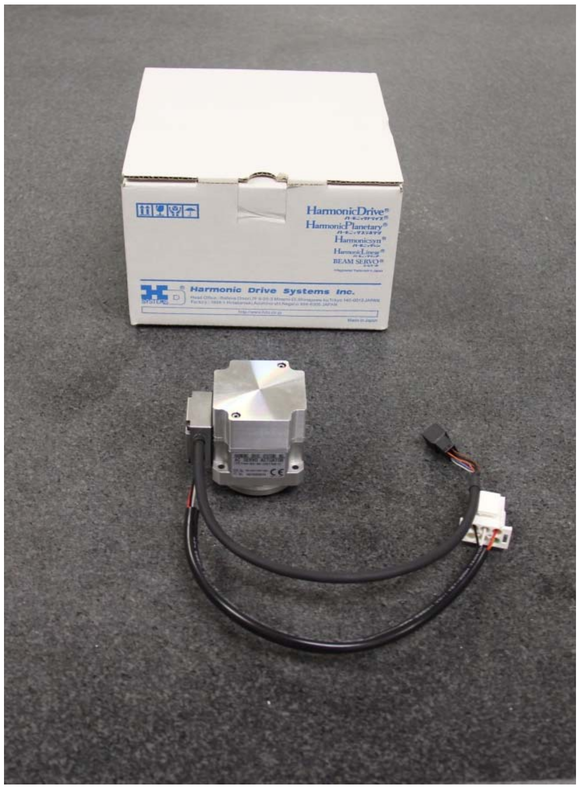


Figure 13. FHA-8C-30-12S17b-E Servomotor.

Figure 14. DEP-090-09 Servomotor Driver. Source: [25].

The driver is programmed via the accompanying Copley Motion Explorer 2 software produced by the Copley Controls company [29]. Driver settings are chosen through the graphic user interface and can be tested through an RS232 serial line connection between the driver and the programming computer. Once settings are satisfactory for the mission requirements, they can be saved and uploaded to the driver's flash memory as a proprietary Copley Controls Axis (.ccx) file. For the .ccx file used in this project and the chosen parameter settings, see Appendix D: MARSMAN_Driver_Load.ccx. Table 1 lists size, weight, power, and cost characteristics of both the servomotor and the driver.

Table 1. Servomotor and Driver Characteristics. Sources: [20], [25], [26].

Hardware	Dimensions	Mass	Input Voltage	Input Current	Cost
Servomotor	50x49x48.5mm	0.5kg	24V DC	3A	$2,500
Driver	196x99x31mm	0.45kg	24V DC	10A	$700

Lesson Learned

To maintain the memory of its current location when the primary power source is removed, the encoder requires input from an alternate power source. In the current design, this alternate power source is a 1.3V Daytona battery. This very important power source was not documented in the initial assembly literature.

b. The FUTEK TFF400 Torque Sensor

The FUTEK TFF400 torque sensor (Figure 15) attaches to the servomotor and senses the amount of torque applied by the servomotor.

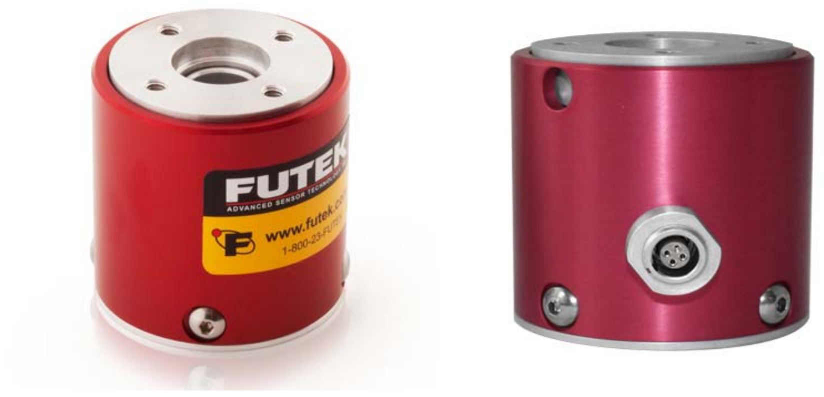


Figure 15. FUTEK TFF400 Torque Sensor, Isometric and Rear Views. Source: [26]

As the motor applies torque to the Link body’s outer plate, the mechanical twisting experienced by the face of the torque sensor is converted into a voltage, which flows into the sensor’s Wheatstone bridge circuit at the +Excitation node (Figure 16). The current signal is read at the +Signal, -Signal, and –Excitation nodes and those currents pass to a load cell amplifier—a COTS integrated circuit that is a new element in the evolved design—and then to the Arduino Due microprocessor for processing and transmission. It should be noted that a low-level torque will produce a reading that is difficult to distinguish from the sensor noise. Table 2 lists size, weight, power, and cost characteristics of the torque sensor.

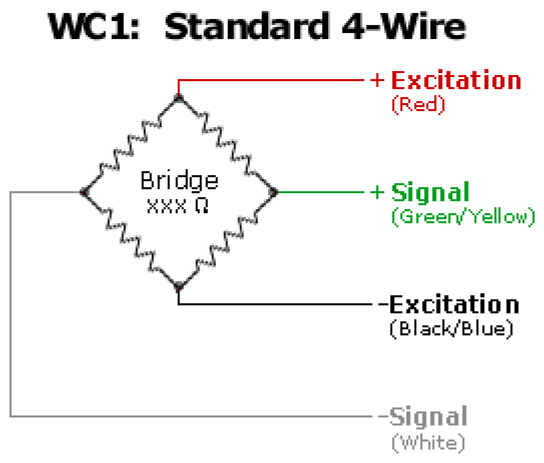


Figure 16. Wiring Diagram for FUTEK's TFF400 Torque Sensor. Source: [30].

Table 2. Torque Sensor Characteristics. Sources: [4], [20].

Hardware	Dimensions	Mass	Output Voltage	Resolution	Cost
Torque Sensor	50.8 x 50.2mm	0.25kg	3mv/V	0.003Nm	$1,040

c. *The Inspired Energy NH2054HD31 Lithium-Ion Battery*

The main power source for each link is the Inspired Energy NH2054HD31 (Figure 17). This rechargeable, lithium-ion battery has a nominal voltage and current life of 14.4V/6.2Ah—an extremely capable battery for this experimentation. The battery and its mounting remained the same from the initial design, but the manner in which the voltage was routed changed to provide the necessary 24V to the driver and servomotor and the necessary 5V to the Arduino.

Figure 17. The Inspired Energy NH2054 Li-Ion Battery. Source: [27].

Lesson Learned

The manufacturer recommends an input voltage for the Arduino Due microprocessor of 7-12V with 6V and 16V being the absolute upper and lower bounds [31]. *Ostensibly, the Arduino Due seems compatible with the NH2054HD31. However, the NH2054HD31 specification sheet notes that the battery can charge up to a maximum of 16.8V—a voltage that will melt the Due's onboard voltage converter* [27]. *Temporary solutions included using slightly discharged batteries and utilizing an auxiliary 9V battery. Ultimately, a voltage regulator was installed between the main power source and the Arduino Due. The discharge curve for this battery is shown in Figure 18.*

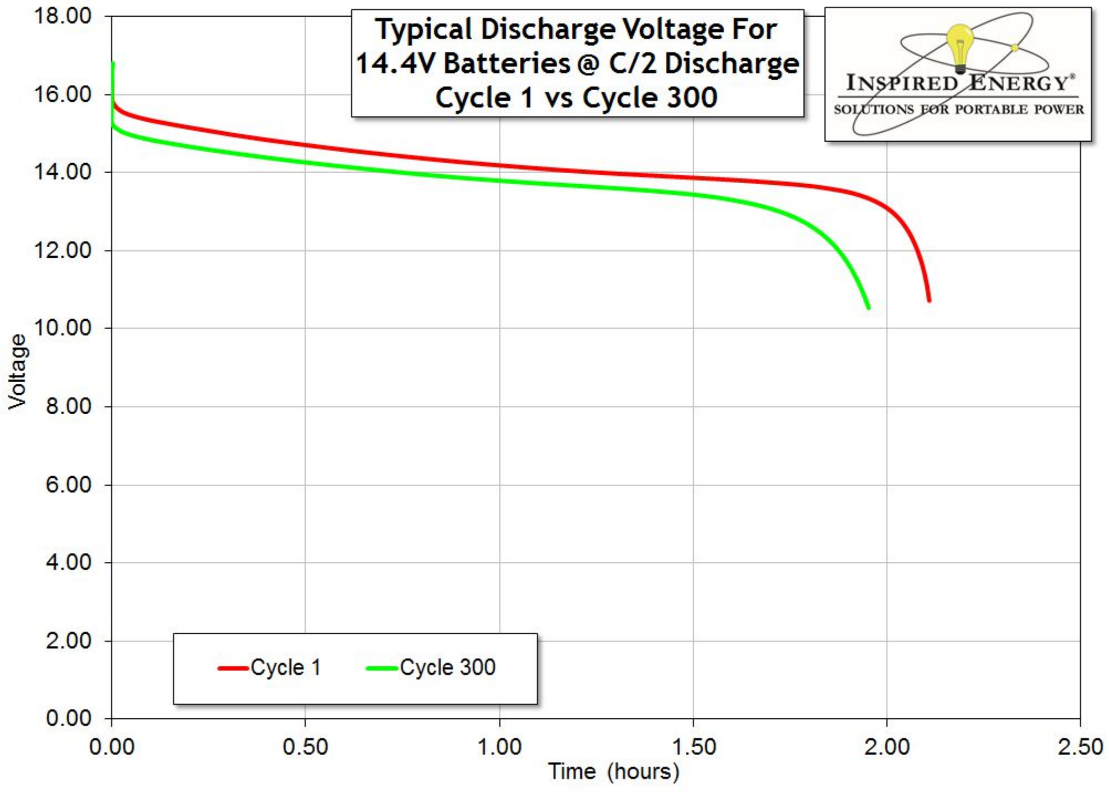


Figure 18. Discharge Curve of 14.4V Li-Ion Battery. Source: [27]

d. The Traco Power 75-2415WI V DC/DC Converter

The Traco Power 75-2415WI V DC/DC converter takes the nominal 14.4V from the main power source and converts it to the 24V needed to power the servomotor driver and the servomotor, which are connected in series. It does not provide any power to the torque sensor as indicated in Link 1’s initial design electrical power architecture [20].

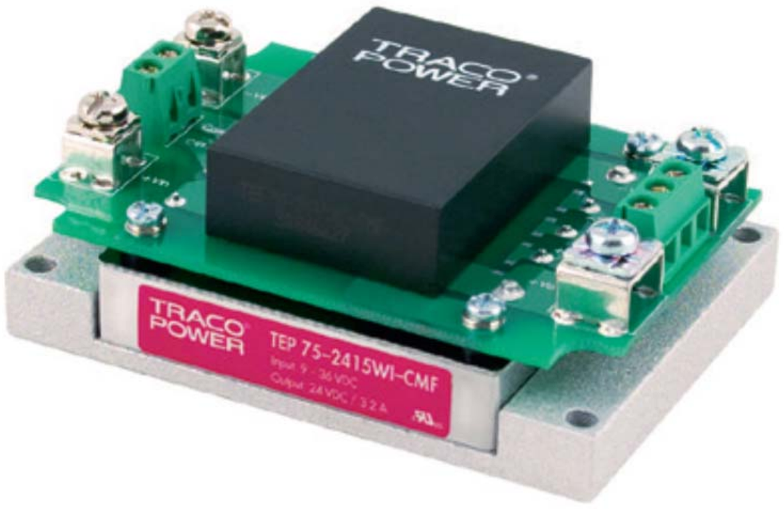


Figure 19. Traco Power 75-2415WI DC/DC Converter. Source: [28].

e. The Arduino Due Microprocessor

The Arduino Due is a hobby-grade microprocessor board that takes advantage of the C programming language and Arduino’s own Integrated Development Environment (IDE) (Figure 20). It is capable of reading the analog torque sensor output signals and converting them to digital signals, which are then passed through a compatible piece of communications hardware. In the initial design, a compatible XBee shield was mated with the Arduino Due to pass position, velocity, and torque data. In the evolved design, the Arduino Wi-Fi shield replaced the XBee shield. The Arduino nominally runs on 3.3V of power but can accept recommended input voltages between 7-12V [31].

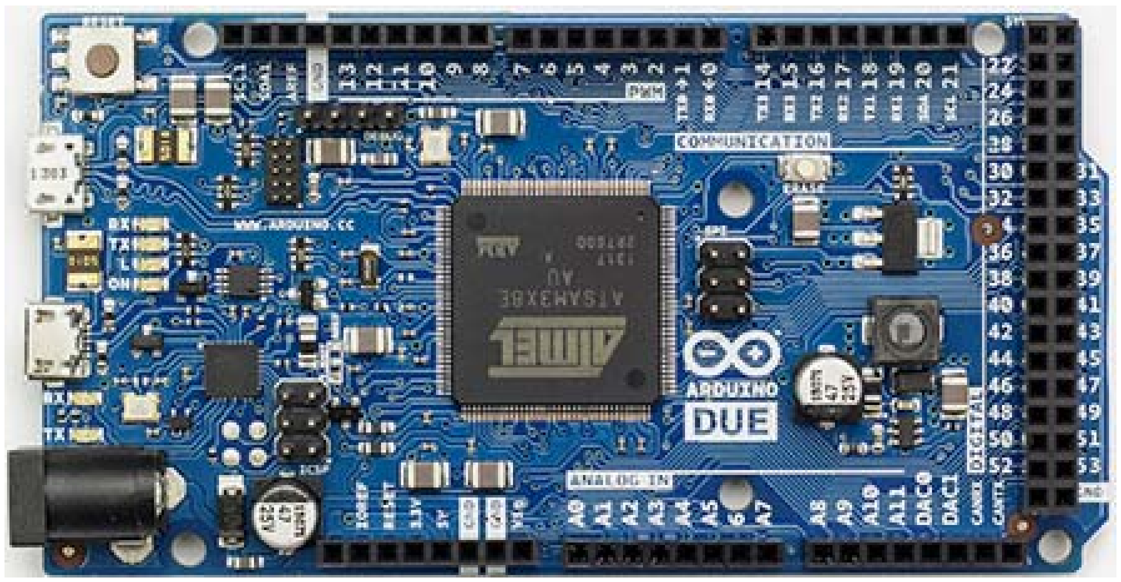


Figure 20. Arduino Due Microcontroller. Source: [31].

f. The LinkSprite RS232 Shield and Serial Connection Hardware

The LinkSprite RS232 shield is an Arduino-compatible shield that allows the Arduino Due to transmit and receive data to/from the servomotor driver via serial cable (Figure 21).

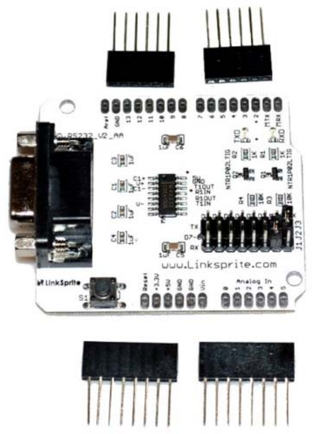


Figure 21. LinkSprite RS232 Shield. Source: [32].

g. The Dantona 3.6V Auxiliary Battery

This Dantona 3.6 battery (Figure 22) provides a bias voltage to the motor encoder when the primary power source, the 14.4V battery, is removed. Without a power supply, the absolute encoder will not retain knowledge of its position.

Figure 22. Dantona 3.6V Battery.

h. Air Delivery System Components

The FSS houses a tank of compressed air, which flows to each of the links via 0.125″ plastic tubing. In keeping with the modular design intent, each link has independent tubing that attaches to the adjacent links via interlocking tubing connectors. A T-splitter allows for air flow to the air bearing itself which connects to the tube via a brass pipe fitting. The bearing is made of a porous type of carbon that allows the compressed air to flow through it. An adjustable threaded ball stud attaches to the bottom of the main link structure and fits within the socket of the bearing. A 3-D printed cap designed and printed in-house keeps the ball joint seated in the bearing. Figure 23 shows these parts.

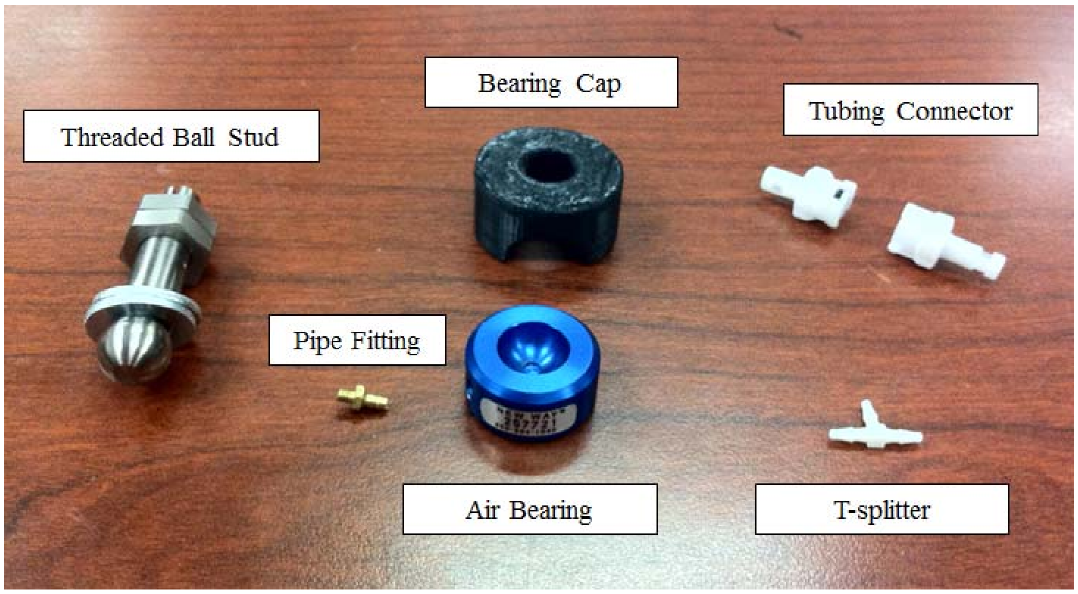


Figure 23. Air Delivery System Components.

3. Hardware Upgrades

With the goals of maximizing COTS equipment to decrease link assembly time, ensure repeatable results, and improve the resolution of the signal reading from the torque sensor, multiple significant hardware changes were implemented. First, the custom circuitry implemented in the original link was replaced with a commercial-off-the-shelf (COTS) load cell amplifier. The removal of the custom circuitry necessitated the installation of a voltage regulator between the DC/DC converter and the Arduino package. Second, the XBee radio transmitter was replaced with an Arduino Wi-Fi shield that was compatible with the Arduino Due board and the originally-used RS232 interface board. Third, the custom serial cable connecting the servomotor driver to the RS232 shield was replaced with functionally identical COTS pieces. Finally, several subtle design improvements were incorporated into the design.

a. The SparkFun Load Cell Amplifier-HX711

To classify the dynamics of the robotic manipulator, information on applied torque is essential. The previously-discussed FUTEK torque sensor is responsible for measuring the torques applied by the motor, but the mechanism by which that data is passed from the torque sensor (an analog signal from a piece of hardware) to analyzable, digital data is an analog-to-digital converter (ADC). In this case, the links employ a Sparkfun load cell amplifier. The SparkFun Load Cell Amplifier-HX711 is shown in (Figure 24).

The crucial component of the load cell amplifier is the HX711 integrated circuit, a chip specifically designed to read a signal from a load cell (i.e. the torque sensor) at high precision. An open-source header file (HX711.h) and a complimentary C++ source code file (HX711.cpp) were available from SparkFun and were incorporated into the final Arduino code (See Appendix A: SparkFun HX711 Load Cell Amplifier Open-source Header File and Appendix B: SparkFun HX711 Load Cell Amplifier Open-source C++ Source Code File).

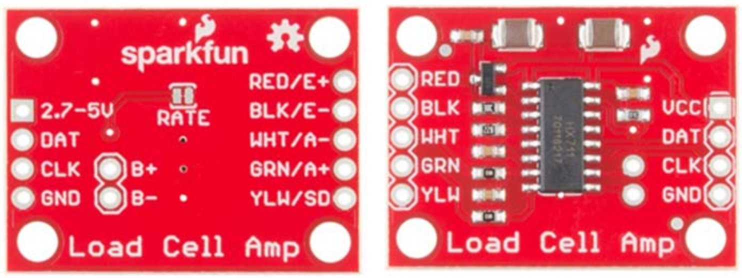


Figure 24. Both sides of the SparkFun Load Cell Amplifier. Source: [33].

To define "high precision," it is necessary to investigate the torque sensor specifications, specifically how it responds to varying voltages. According to its specifications, the torque sensor has a rated output of 2mV/V when the torque input is at a maximum (400 in-oz or approximately 2.82 Nm) [26]. With the 15V excitation to the torque sensor from the DC/DC converter in the original design, the voltage difference between the terminals of the torque sensor was found to vary by a very small amount, only $\pm$30mV [20]. The intent of the custom circuit board in that design was to boost the strength of the torque sensor output, ideally to the 3.3V accepted by the Arduino's analog input port [20]. The Arduino itself can only serve as a 10-bit analog-to-digital converter (ADC). Thus, each torque sensor reading can be represented by one of ten integers from zero to 2^{10} (or 1,024). While this method was workable, it only provided a best-case resolution of $3.3\text{V}/2^{10} = 3.2\text{mV}$, which is a poor resolution in relation to the $\pm 30\text{mV}$ measured variation. Implementing a COTS load cell amplifier eliminated the need for custom parts while allowing for greater resolution of the torque signal.

The Spark Fun load cell amplifier uses the HX711 24-bit analog-to-digital converter (ADC) to transform the torque sensor's analog electrical signal output into a digital signal that can be processed by the Arduino stack and analyzed within Simulink/MATLAB. Since the HX711 has a capacity of 24 bits, it can represent 2^{24} (or 16,777,216) integers. The nominal middle of the HX711's range is 8,388,608. With zero load the torque sensor read in the range of 8,374,000 with fluctuations due to sensor noises—a reading reasonably close to the nominal middle. With 24 bits, the best-case resolution becomes $3.3\text{V}/2^{24} \approx 1.97\times10^{-4}\text{mV}$, which is a vast improvement over using the Arduino's ADC.

The second step in the calibration of the data was to calculate the analog input to the torque sensor. With the Arduino putting out 5V, the analog voltage leaving the HX711 (pin 3, Analog Voltage Drain or "AVDD"), can be calculated via the equation provided in the system diagram, Equation 1.

$$AVDD = \frac{VBG(R1 + R2)}{R1} \tag{1}$$

In the above equation, the bandgap voltage (VBG) is the reference bypass analog output voltage (pin 6), a fixed value of 1.25V. R1 and R2 are the resistor values within the circuit, $8.2\,k\Omega$ and $20\,k\Omega$, respectively [34]. Note that the R1 and R2 values listed in the product diagram are reversed—a mistake that caused some confusion during initial measurements. When calculated as listed in the diagram, the analog power supply is found to be 1.78V—outside the nominal range of the HX711 (2.2 to ~5.5V). The voltage was measured at the input to the torque sensor and found by voltmeter to be 4.33V, very close to the mathematically predicted AVDD of 4.30V when R1= $8.2\,k\Omega$ and R2=$20\,k\Omega$.

On the HX711, Channel A can be programmed with a gain of 128 or 64 [34]. The higher gain value was chosen and implemented via the Arduino code to allow for a higher resolution in the sensor output. The system specifications state that a gain of 128 and an AVDD of 5V, the corresponding voltage output is expected to be within the range of +-20mV [34]. In the case of the torque sensor, the AVDD value is less than 5V, so the actual range can be calculated by Equation 2 to be ± 17mV.

$$\frac{\text{Nominal Range}}{\text{Nominal AVDD}} = \frac{\text{Actual Range}}{\text{Actual AVDD}} \tag{2}$$

Finally, the torque sensor itself has a calibrated output of 1.9873mV/V at 400 in-oz (2.8246Nm). With 4.3V actually applied to the sensor, the torque constant (k) as a function of voltage can be calculated. By Equation 3, every mV applied results in .33Nm of torque. This mathematical model was implemented in the Arduino code. The results of the code were verified by measuring the applied torque with an analog torque watch and the voltage output with a voltmeter.

$$k = \frac{2.8246Nm}{1.9873\,Nm/mV \times 4.3V} = .33\,Nm/mV \tag{3}$$

b. The Arduino Wi-Fi Shield

The Arduino Wi-Fi shield replaced the Xbee wireless communication shield from the original design for a number of reasons. The Xbee shield was capable of adequate transmission rates and ranges for the application, but it depended upon the Zigbee protocol, a more obscure cousin of Wi-Fi. Since the FSS and the VICON system use Wi-Fi, an Arduino-compatible Wi-Fi shield seemed like the obvious choice for implementation although Bluetooth was briefly considered. Second, communication with the on-board XBee required attaching an XBee transmitter to the USB port of a command laptop. A Wi-Fi shield is capable of interfacing directly with the command laptop's organic Wi-Fi capability, and although an external router was used in this thesis, a more capable microprocessor (for example, a Raspberry Pi) could be used to create an *ad hoc* Wi-Fi network. Finally, due to a lack of on-hand supplies and the discontinuance of the Xbee shield, determining a replacement became necessary. In an ironical twist, the Arduino Wi-Fi shield that was chose as the successor to the Xbee shield was itself discontinued by Arduino during the evolutionary design. A top view of the Arduino Wi-Fi shield is shown in Figure 25.

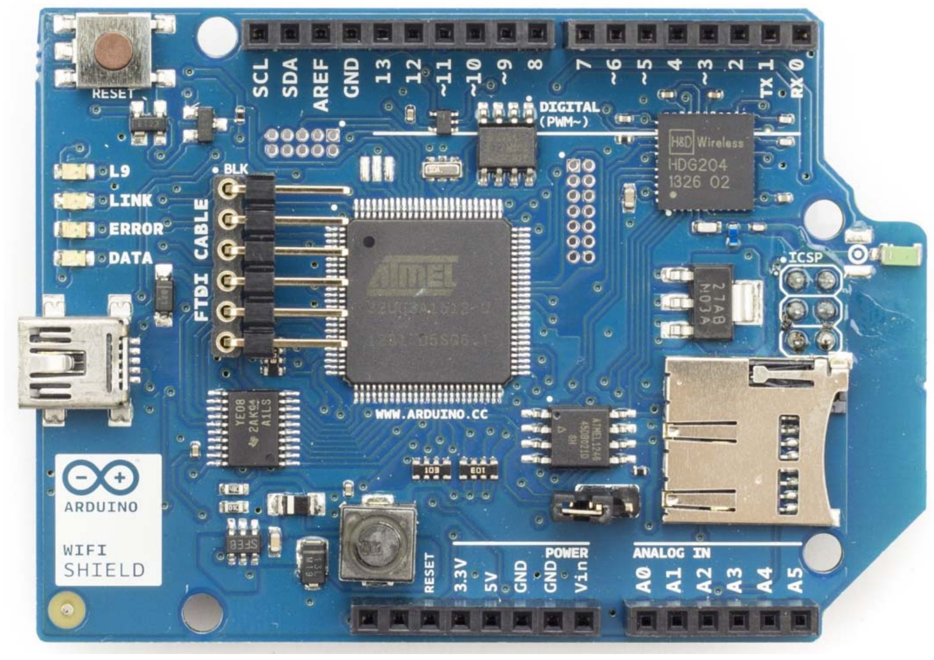


Figure 25. Arduino Wi-Fi Shield. Source: [35].

The final limitation of the Arduino Wi-Fi shield is that it is no longer being manufactured; Arduino has discontinued the product line and is no longer updating the Wi-Fi Shield webpage. Thus, as efforts were being made to integrate the shield into the existing configuration, the hardware itself was already obsolete. Future work must explore the integration of a replacement for Arduino Wi-Fi shield, such as the Raspberry Pi microcontroller with Wi-Fi dongle (more on that in Chapter VI).

c. The RS232 Shield and Serial Cable Connection

The RS232 shield requires a gender changer to connect the serial cable to the servomotor. Both ends of the serial cable are capped with RJ12 connectors. The introduction of the COTS gender changer and a new serial cable that follows the U.S. Bell System coloring scheme eliminated the need for tedious customization of a serial connector and the use of a null modem.

Lesson Learned

Not all serial cables are wired the same. The cables used in this project use the U.S. Bell System Coloring scheme. As viewed from the front of the RJ12 connector, the colors of the wires are (from left to right) white, black, red, green, yellow, blue. When both ends of the cable are placed side-by-side, they display the same pattern (the" standard" configuration). If the color patterns mirror one another (the "reversed" configuration), the cable still can be used, but a null modem needs to be inserted between the RJ12 connector and the RS232 port to flip the transmit and receive signals.

d. Voltage Regulator

To keep the battery's output voltage in a safe operating range for the Arduino, a voltage regulator was installed to convert the 16.8V maximum voltage to a 5V input for the microprocessor (Figure 26).

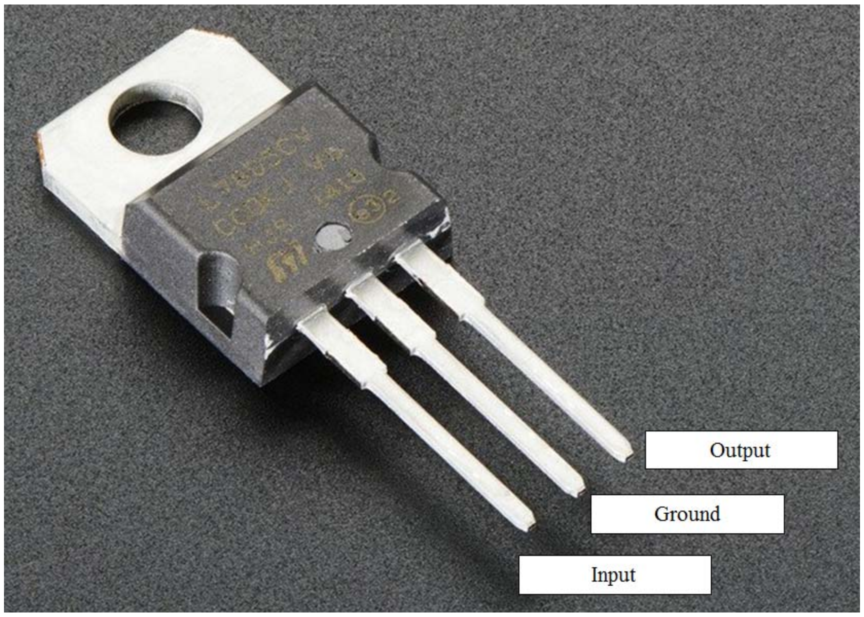


Figure 26. Voltage Regulator. Adapted from [36].

e. Miscellaneous Modifications and Improvements

In addition to the significant component changes, several more subtle changes were included in the design. Mounting hole placements were modified to better accommodate placement of the microcontroller, a Velcro strap was added to prevent rattling of the battery during testing, and bolt usage was standardized to the greatest extent possible. Apart from the bolts required to mount the torque sensor, all other bolts are button hexagonal head, 4mm diameter.

Finally, a link with an attached air bearing presented a problem: the link could not stand because the air bearing was jutting out the bottom of the structure. Initially, cardboard boxes were used to prop up the completed links, but these were fairly unstable (and unsophisticated). To address that shortcoming, the freely available Microsoft 3D Builder application was used to model "shoes" for the links. The shoes (Figure 27) lift the link off of the table high enough so that the dangling air bearing does not touch. As a side note, the 3D Builder software was also useful for viewing the link components when licenses with the more advanced Computer Aided Drafting Software became problematic.

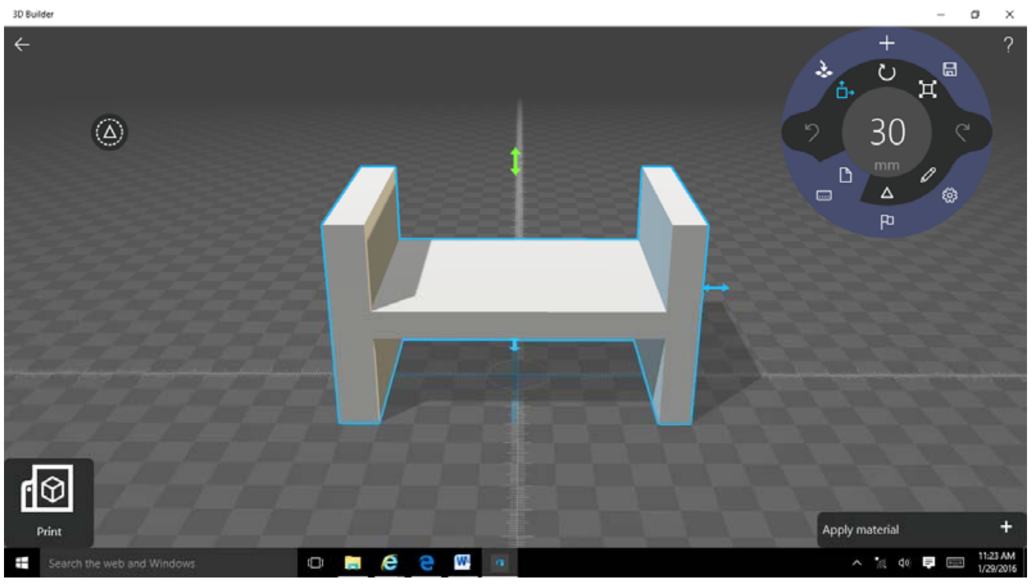


Figure 27. “Shoe” Developed Using the Free Microsoft 3D Builder Application

III. ASSEMBLY DOCUMENTATION

A. INTENT OF THE ASSEMBLY DOCUMENTATION SECTION

The intent of this section is to document the assembly of a link to a sufficient degree that future researchers wishing to replicate the experimental campaign, evolve the present design, or develop an entirely different robotic manipulator design have a well-documented, successful design from which to begin their work. Although assembly documentation exists prior to this, given the hardware upgrades, a newer, more thorough documentation is necessary. This documentation additionally includes thorough labeling of part names, wiring diagrams that will prevent future agonizing over pin-out schematics and hardware interfaces, an introduction to the driver programming software interface, and an in-depth discussion of assembly methodologies and lessons learned. In truth, no two links were constructed in exactly the same sequence. However, by the completion of Link 4, a best-practice approach was established.

At the outset of the construction of each Link, the idea to transform the side walls of the link structure into slide-out walls resurfaced. Walls that could slide out would avoid the difficulties of fitting large hands in small places and trying to tighten nuts onto inconveniently located bolts. The sliding wall modification was never attempted for two reasons. First, in the interest of research focus, a time-consuming diversion into CAD modifications was not desirable. Second, and more importantly, it was desirable to keep all of the links as structurally similar as possible to simplify the initial assessment of the system kinematics and dynamics. It is possible that sliding walls would have diminished the rigid-body behavior of the manipulator link.

Previous documentation of the link assembly favored the approach of first mounting the hardware to the structure and wiring throughout the process. I suggest that a simpler method is to complete all of the subsystem wiring first. Broadly, the manipulator link can be viewed as four subassemblies: the torque sensor/servomotor subassembly, the circuit board subassembly, the power subassembly, and the servomotor driver (which simply mounts to the wall of the main link structure). Thus, the documentation of the assembly begins not with the link structure but with the preparatory component wiring.

B. BUILDING A LINK

1. The Servomotor Cable Connectors

The servomotor has two wiring outputs, the motor leads (top cable in Figure 28) and the encoder cable (bottom cable in Figure 28). Neither of these two cables comes with a connector that is compatible with the servomotor driver. Figure 32 shows the motor with stock connectors; the motor leads connector is the white connector, and the encoder cable connector is the black connector.

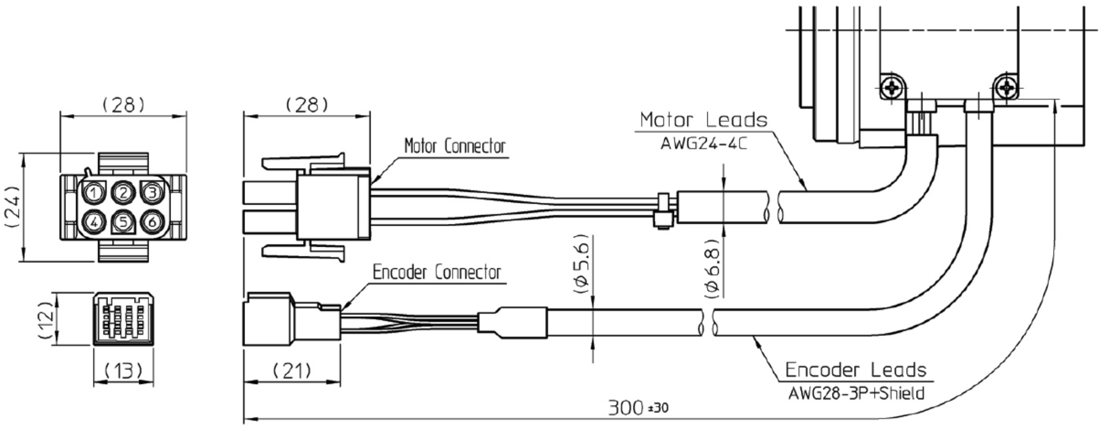


Figure 28. Servomotor Cabling Schematic. Source: [25].

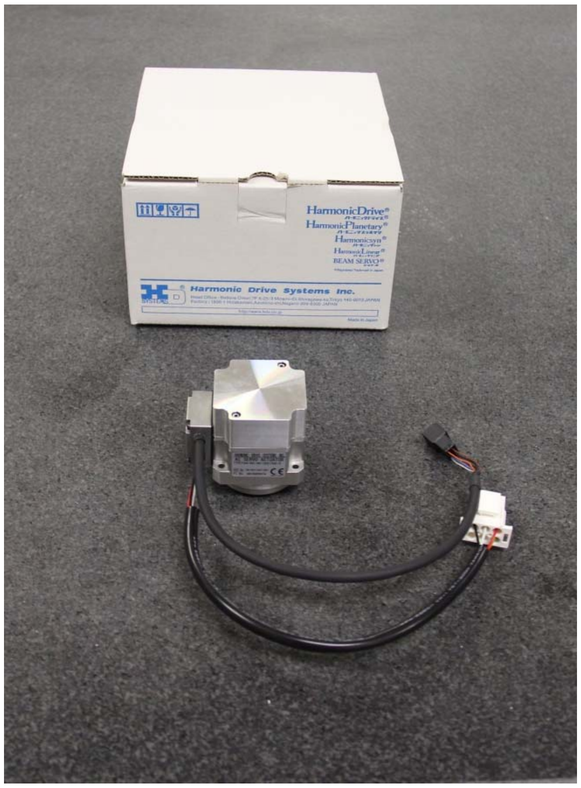
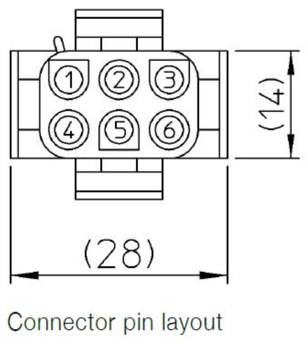


Figure 29. Servomotor with Stock Connectors.

The motor leads are initially attached to the six-pin connector shown in Figure 33. The red, white, and black wires correspond to the power inputs for the three-phase motor (phases U, V, and W, respectively). At continuous operation, the driver provides 2.3A, 2.4A, and 1.8A of current via those three lines (U, V, and W, respectively) [37]. Pin 4 attaches to the green/yellow wire, which serves as the ground. The designation of the green/yellow wire as "PE" in Figure 30 stands for "Protective Earth," a term used by the International Electrotechnical Commission [38].



Connector model: 350715-1

Pin model: 3506901 (E: 770210-1)

Manufactured by AMP

Pin No.	Color	Motor lead
1	Red	Motor phase-U
2	White	Motor phase-V
3	Black	Motor phase-W
4	Green/yellow	PE
5	-	-
6	-	-

Figure 30. Motor Lead Connector Layout (Units in mm). Source: [39].

The stock connector is removed and replaced with a 4-position J2 connector (that is, it connects to the J2 port on the servomotor driver). This connector, along with the servomotor encoder cable connector (discussed in the next section) comes in a connector kit available from Copley Controls Corporation (kit name: AEP-CK, AMP-CK, part number: 84-00147-000 Rev A). The new motor lead connector's serial number and nomenclature are "57-00605-000: Plug, RoHS, Euro-Style, 4 position, 5.08 mm." The colored wires are attached to the new motor lead connector in accordance with Figure 31.

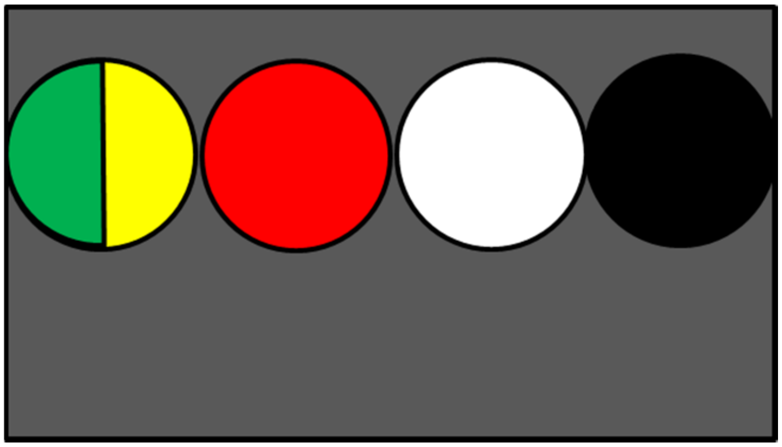
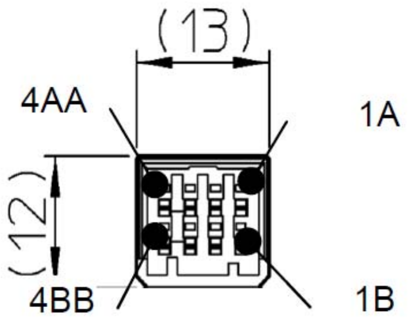


Figure 31. Motor Cable Wiring with 4-Position J2 Connector.

2. The Encoder Cable Connector

The encoder cable connects to the motor driver and reads position data. From position data, velocity data is derived on-board the driver. The pinout of the stock connector and corresponding descriptions of the pins is shown in Figure 32.



Connector pin layout

Connector model: 1-1903130-4
Pin model: 1903116-2 or 1903117-2
Manufactured by AMP

◆ Encoder cable (US 200)

Pin No.	Color	Signal	Remarks
1A	White	Vcc	Power supply input +5V
1B	Black	GND (Vcc)	Power supply input 0V (GND)
2A	Blue	SD+	Serial signal differential output (+)
2B	Purple	SD-	Serial signal differential output (-)
3A	–	No connection	–
3B	Shield	FG	Frame Ground
4A	Orange	Vbat	Battery +
4B	Brown	GND (bat)	Battery – (GND)

Figure 32. Pinout of Encoder Cable with Stock Connector. Source: [39].

The 20-pin J3 feedback connector (57-00608-000: Connector, RoHS, 20 position, Mini-D, solder cup) replaces the stock connector. The plastic shell (47-00130-000: Backshell, RoHS, plastic, 20 position, Mini-D) encases the new 20-pin connector. Figure 33 shows a picture of the new connector pieces alongside the encoder cable.

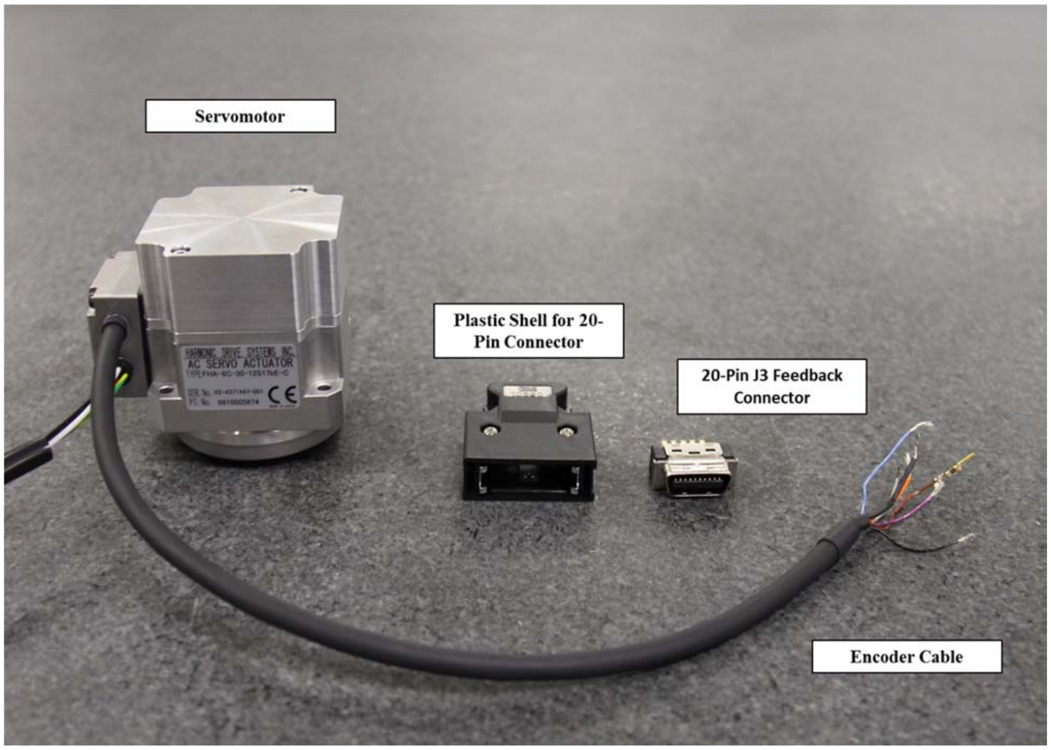
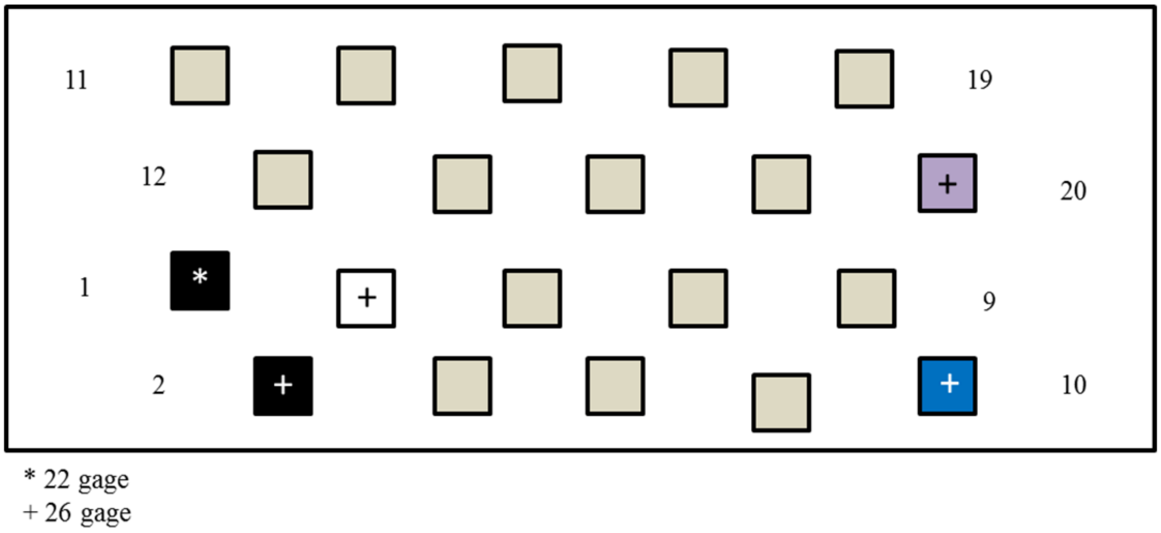

Servomotor
Plastic Shell for 20-Pin Connector
20-Pin J3 Feedback Connector
Encoder Cable

Figure 33. Replacement Connector Pieces Alongside the Encoder Cable.

The pinout diagram for the 20-pin connector is shown in Figure 34. Notice that the brown and orange wires (the auxiliary battery negative and positive connections, respectively) are not connected to a pin. These wires attach to a separate power source, a 3.6V Dantona battery, as shown in Figure 35. This 3.6 battery provides power to the motor encoder when the primary power source, the 14.4V battery, is removed. Because it is an absolute encoder, the encoder will not retain knowledge of its position without a power supply.



For color codes, reference Figure 32. Gold squares represent unused pins.

Figure 34. Pinout of Servomotor Encoder Cable with 20-Pin Connector.

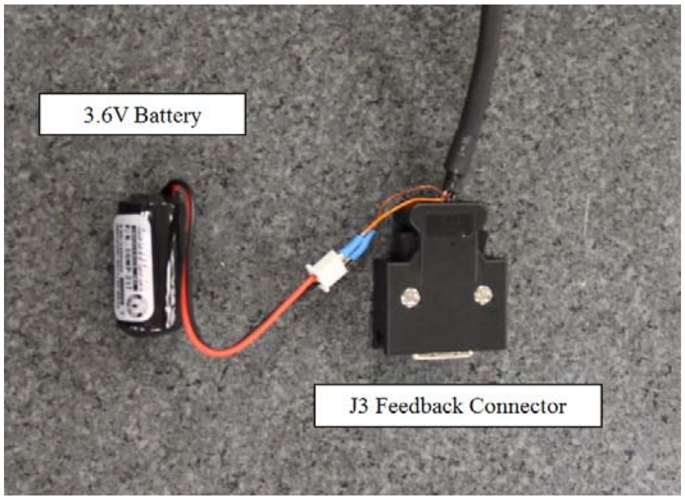


Figure 35. Encoder Cable with 20-Pin Connector and Attached Battery.

3. The Torque Sensor Wiring

As discussed in the review of hardware, the FUTEK Model TFF400 torque sensor has a four-pin output receptacle. The mating connector for this receptacle, the FGG.0B.304.CLAD35 type connector, is not included as a component of the torque sensor but is purchased separately. Wires were soldered to the four-pin connectors, and the shell of the connector reassembled around the pins. The .CLAD35 connector has a red dot which aligns with the torque sensor receptacle. The output wires were labeled (+EXEC, -EXEC, +SIG, -SIG) to avoid confusion during their connection to the load cell amplifier connections of the same names. Figure 36 shows the torque sensor with mated .CLAD35 connector. Although no standard wiring scheme was used from link to link, in the image below, white wires attach to the load cell amplifier, and the green wires will eventually connect to the Arduino assembly.

Lesson Learned

The red dot on the shell of the .CLAD35 connector swivels about the four-pin connector itself; it is not fixed to the four-pin connector. Thus, an alignment of the connector's red dot with the torque sensor's red dot does not necessarily mean that the intended wires are aligned with the intended receptacles. Extra care should be taken to ensure that the output of each wire is known so that it can be connected to the appropriate position on the load cell amplifier. In one instance where the lead wires were mislabeled coming out of the receptacle, no negative effects on the hardware were noted.

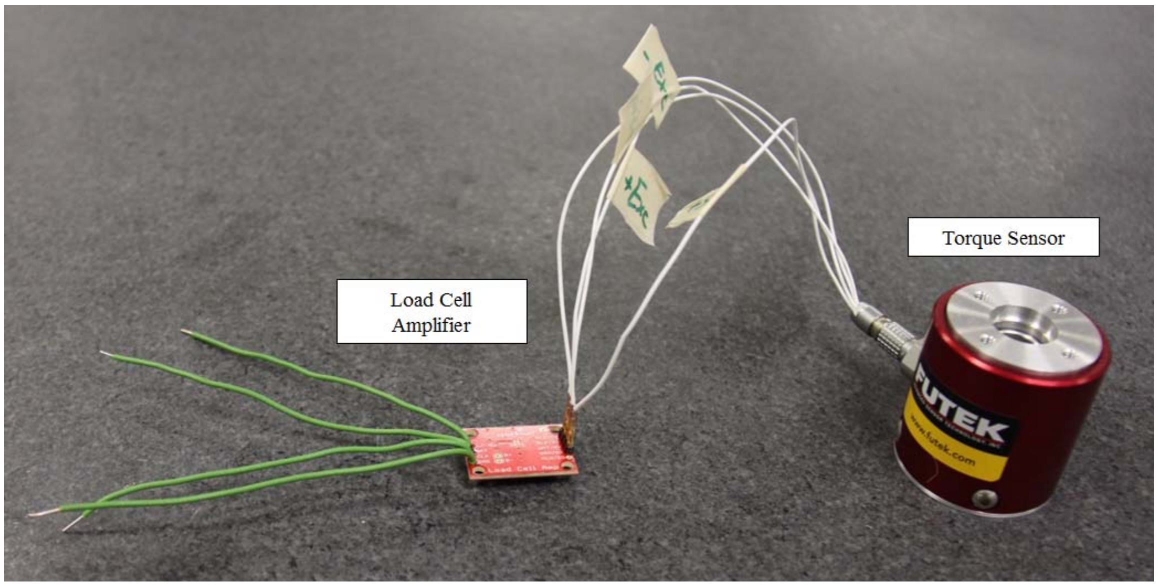


Figure 36. Torque Sensor Wired to the Load Cell Amplifier.

4. Battery Connector and DC/DC Converter

From the nominally 14.4V Li-ion battery, power flows according to the block diagram in Figure 37. The DC/DC converter connects to the power and ground outputs of the battery terminal. The driver and motor require 24V, so the voltage is up-converted and passed to the driver's J1 connection portal via a three-pin connector similar to the one used on the motor cable wiring (from the same connector kit, part 57-00604-000: Plug, RoHS, Euro-Style, 3 position, 5.08 mm). Note that only the voltage and ground pins are used in the three-pin connector.

While the driver/motor assembly requires 24V, the voltage going into the Arduino assembly needs down-converted. This is managed via a Low Dropout regulator (LDO) that outputs the necessary 5V. The LDO's input of 14.4V (nominal) comes directly from the battery. Figure 37 shows a block diagram of the electrical power architecture.

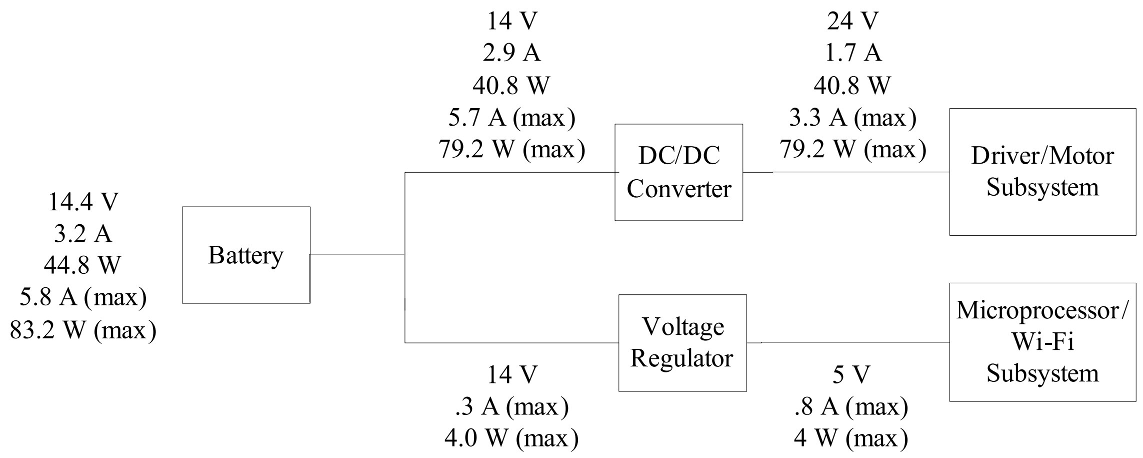


Figure 37. Block Diagram of the Electrical Power Architecture.

5. The Arduino Subassembly

The Arduino subassembly consists of the Arduino Due microcontroller, the Arduino Wi-Fi Shield, and the RS232 shield. The Due uses a universal asynchronous receiver/transmitter (UART) microchip to transmit data via the transistor-transistor logic (TTL) method, which for the Due remains at a voltage level between zero and 5V [40]. The RS232 shield uses the power from the Due and converts the data from TTL to the

Recommended Standard 232 (RS232) format, which is compatible with the driver. From a hardware perspective, the components are simply stacked on top of one another: the Wi-Fi shield plugs into the Due, and the RS232 shield plugs into the Wi-Fi shield. The RS232 shield connects to the motor driver via a serial cable. As noted previously, the proper serial cable (i.e., a serial cable that uses the standard U.S. Bell System coloring scheme) eliminates the need for a null modem but requires a gender changer to align the pins. Jumper wires are used to connect the Due's transmit and receive ports to the RS232's transmit and receive ports. Given the capability of the Due, different transmit/receive port combinations are possible.

After examining the Due pinout diagram and experimenting with different combinations, two successful combinations are offered here. Thus, even with a faulty board or inexplicable malfunction, one combination or the other should allow for a viable solution. In the first successful case, the Due's port 16 (Tx) is connected to the RS232 shield's digital port 3, and the Due's port 17 (Rx) is connected to the RS232 shield's digital port 2. This configuration works when the jumper connectors on the RS232 shield connect J2 and J3 in the D2 row and J1 and J2 in the D3 row. In the second case, the Due's ports 17 (Rx) and 16 (Tx) are connected to the RS232 shield's digital ports 5 and 6, respectively. This configuration requires that the jumper connectors on the RS232 shield connect J2 and J3 in the D5 row and J1 and J2 in the D6 row. The connection of the Arduino assembly to its power supply and to the load cell amplifier is explained in section 5 with the discussion of the Arduino subassembly mounting.

6. Mounting the Components to the Structure

With the majority of the wiring completed, the subassemblies can be joined to the outer structure. Mounting the parts may seem a straightforward task, but the order in which they are mounted is important. Through the experience of building multiple links, an order for assembling the components was decided upon that allowed for necessary component testing while avoiding self-interference. It is, for example, not advisable to mount the Arduino assembly until near the end of the assembly because trouble-shooting the power subsystem may be necessary, and accessing the DC/DC converter or battery mount is difficult if the Arduino assembly is mounted. The procedure outlined below incorporates similar lessons learned.

First, the torque sensor is screwed to the motor carriage with a piece of rubber matting in between them. The rubber matting was a carry-over from the initial design and is not strictly necessary for the link function. The rubber does, however, help to seat the torque sensor snugly on the motor carriage and may possibly reduce the transfer of the motor's vibration to the torque sensor (Figure 38). The pin connector with completed wiring (including its attachment to the load cell amplifier) is inserted.

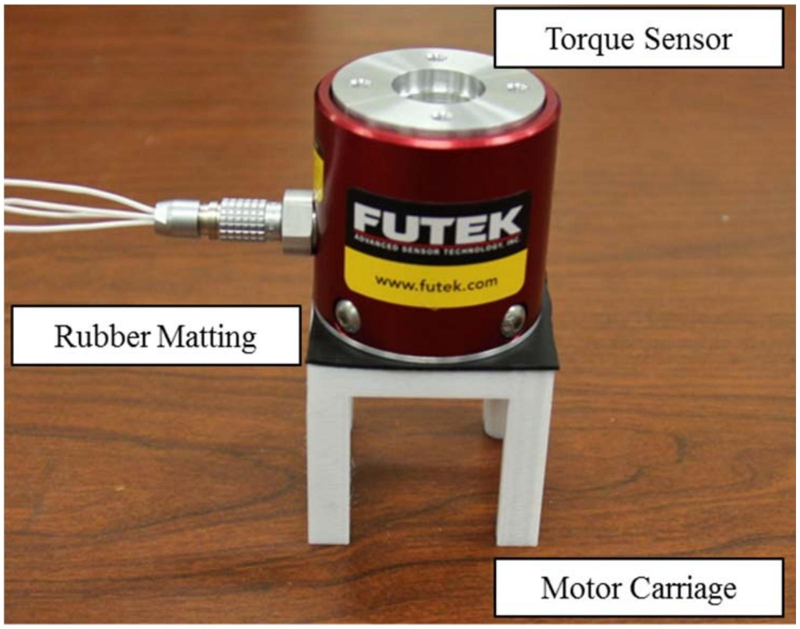


Figure 38. Torque Sensor Mounted on the Motor Carriage.

Next, the servomotor is mounted to the outer plate (Figure 39). The servomotor cables will not run through the slot in the outer plate but through the slot in the link structure.

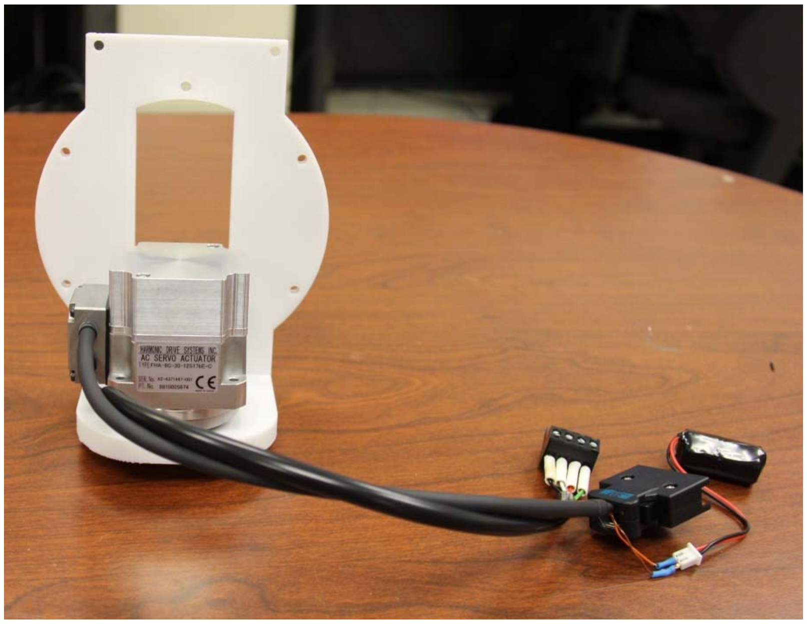


Figure 39. Servomotor Mounted to Outer Plate

The motor carriage with affixed torque sensor is slid down over the servomotor itself, fitting snugly in place (Figure 40).

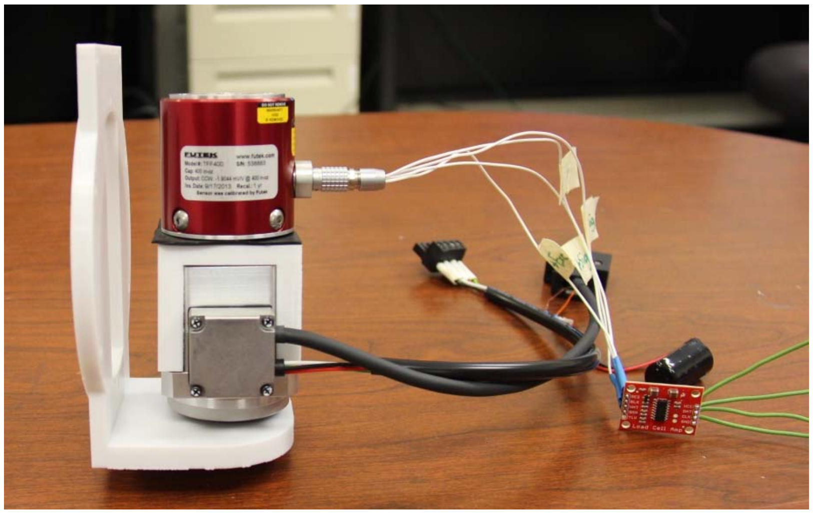


Figure 40. Mated Torque Sensor and Servomotor

Two circular bearings (Figure 41) are added, one to the axel on the bottom of the outer plate and a second on the axel of the outer bracket. The inner bracket is screwed to the top of the torque sensor. Once affixed to the outer plate, the outer bracket's bearing fits inside of the circular hole on the top of the inner bracket. Figure 42 shows the completed joint assembly. The cables are run through the opening in the link structure, and the inner bracket is bolted to the link structure.

Figure 41. Circular Bearings for Use on Outer Plate and Outer Bracket.

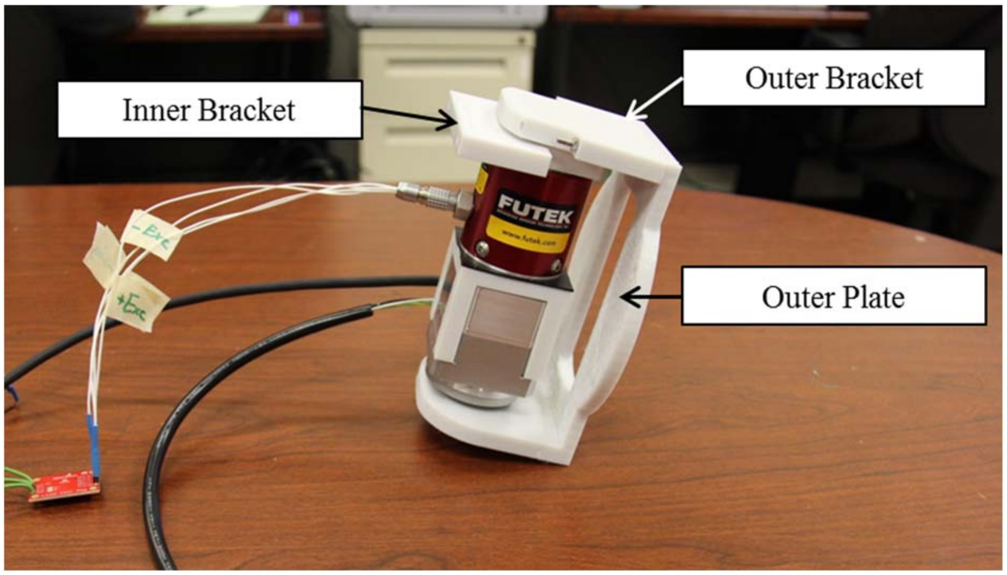
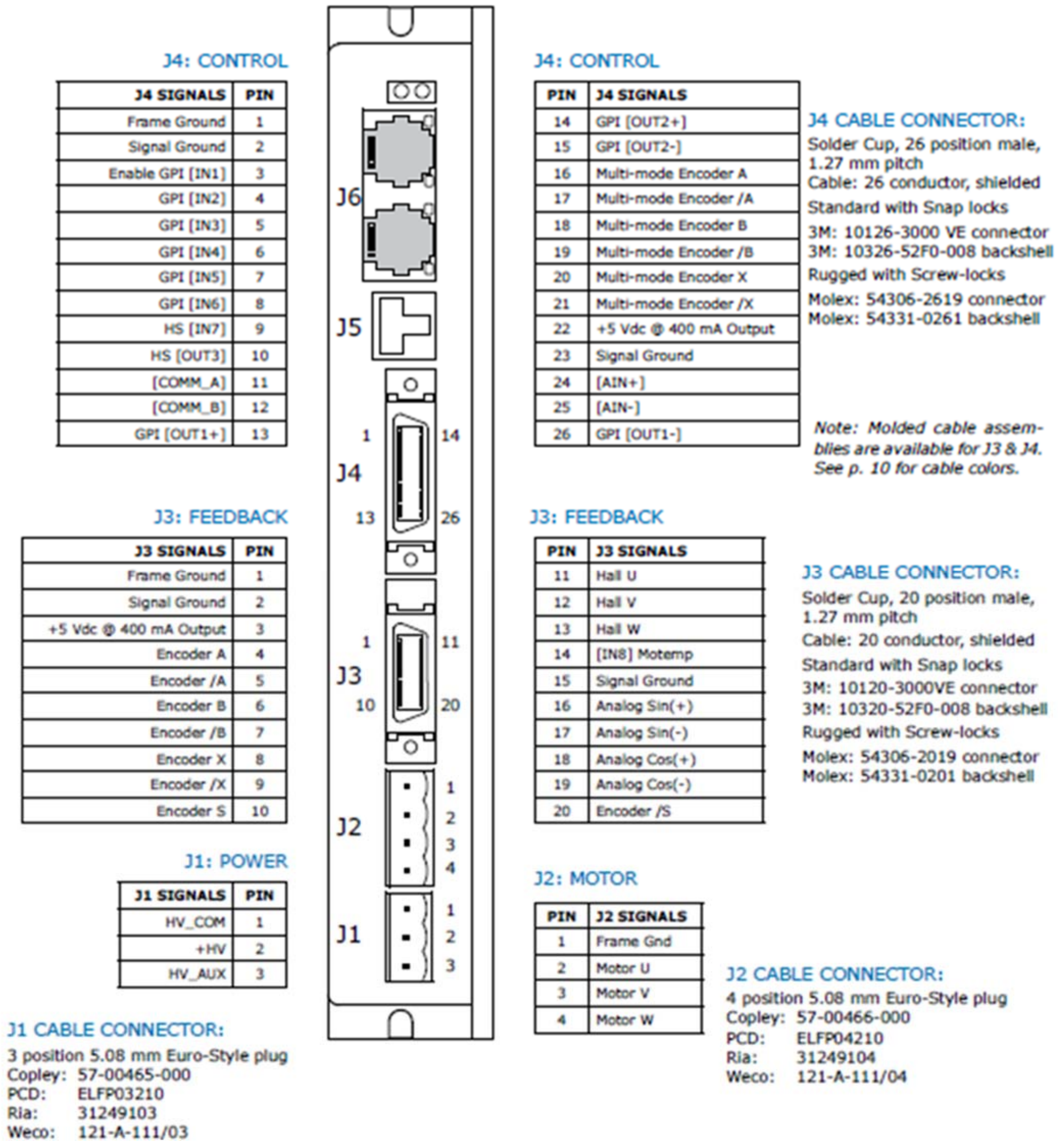


Figure 42. The Completed Joint Assembly.

With the joint assembly complete, the servomotor driver can be bolted to the sidewall of the link structure. It will be programmed when the power supply is operational. The battery connector is bolted to the floor of the link structure, and the DC/DC converter is mounted to the link structure's wall opposite the servomotor driver. The three-pin power connector from the DC/DC converter is connected to the J1 receptacle of the servomotor driver, the motor cable's four-position connector is plugged into the driver's J2 receptacle, and the servomotor encoder with 20-position connector is plugged into the driver's J3 receptacle. The J5 receptacle is used for the connection of the serial cable to the RS232 shield. The J4 and J6 ports are not currently used. Figure 43 shows a diagram of the driver connection ports, and Figure 44 shows the link assembly as it looks at this stage in the construction.



J4: CONTROL

J4 SIGNALS	PIN
Frame Ground	1
Signal Ground	2
Enable GPI [IN1]	3
GPI [IN2]	4
GPI [IN3]	5
GPI [IN4]	6
GPI [IN5]	7
GPI [IN6]	8
HS [IN7]	9
HS [OUT3]	10
[COMM_A]	11
[COMM_B]	12
GPI [OUT1+]	13

J4: CONTROL

PIN	J4 SIGNALS
14	GPI [OUT2+]
15	GPI [OUT2-]
16	Multi-mode Encoder A
17	Multi-mode Encoder /A
18	Multi-mode Encoder B
19	Multi-mode Encoder /B
20	Multi-mode Encoder X
21	Multi-mode Encoder /X
22	+5 Vdc @ 400 mA Output
23	Signal Ground
24	[AIN+]
25	[AIN-]
26	GPI [OUT1-]

J4 CABLE CONNECTOR:
Solder Cup, 26 position male, 1.27 mm pitch
Cable: 26 conductor, shielded
Standard with Snap locks
3M: 10126-3000 VE connector
3M: 10326-52F0-008 backshell
Rugged with Screw-locks
Molex: 54306-2619 connector
Molex: 54331-0261 backshell

Note: Molded cable assemblies are available for J3 & J4. See p. 10 for cable colors.

J3: FEEDBACK

J3 SIGNALS	PIN
Frame Ground	1
Signal Ground	2
+5 Vdc @ 400 mA Output	3
Encoder A	4
Encoder /A	5
Encoder B	6
Encoder /B	7
Encoder X	8
Encoder /X	9
Encoder S	10

J3: FEEDBACK

PIN	J3 SIGNALS
11	Hall U
12	Hall V
13	Hall W
14	[IN8] Motemp
15	Signal Ground
16	Analog Sin(+)
17	Analog Sin(-)
18	Analog Cos(+)
19	Analog Cos(-)
20	Encoder /S

J3 CABLE CONNECTOR:
Solder Cup, 20 position male, 1.27 mm pitch
Cable: 20 conductor, shielded
Standard with Snap locks
3M: 10120-3000VE connector
3M: 10320-52F0-008 backshell
Rugged with Screw-locks
Molex: 54306-2019 connector
Molex: 54331-0201 backshell

J1: POWER

J1 SIGNALS	PIN
HV_COM	1
+HV	2
HV_AUX	3

J1 CABLE CONNECTOR:
3 position 5.08 mm Euro-Style plug

Copley:	57-00465-000
PCD:	ELFP03210
Ria:	31249103
Weco:	121-A-111/03

J2: MOTOR

PIN	J2 SIGNALS
1	Frame Gnd
2	Motor U
3	Motor V
4	Motor W

J2 CABLE CONNECTOR:
4 position 5.08 mm Euro-Style plug

Copley:	57-00466-000
PCD:	ELFP04210
Ria:	31249104
Weco:	121-A-111/04

Figure 43. Diagram of the Harmonic Drive Connection Ports. Source: [41].

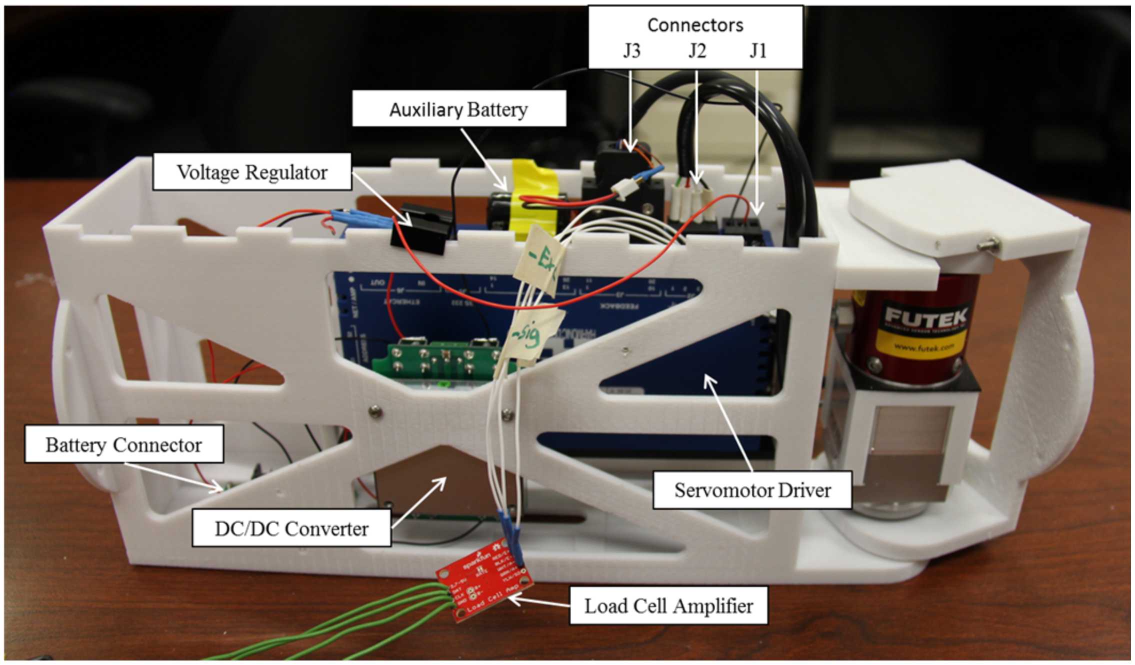


Figure 44. Link with Joint Assembly, Mounted Servomotor Driver and DC/DC Converter

At this stage, the main power supply (the 14.4V Li-ion battery) can be inserted and the servomotor driver programmed. If the wiring is done correctly, the power indicator light (the AMP light) on top of the driver will light solid green. Attaching a serial cable between the J5 (RS232) port on the driver and a computer with the Copley Motion Explorer 2 (CME2) software allows for configuring the motor driver. Figure 45 shows the CME2 graphic user interface (GUI) upon initialization of the software.

As mentioned in the hardware section, the required configuration file is a Copley Controls Axis (.ccx) file. Through the GUI the user can manipulate system parameters and create a unique ccx file, which is saved using the yellow floppy disk icon near the top of the GUI window. For the ccx file used in this project and the chosen parameter settings, see Appendix D: MARSMAN_Driver_Load.ccx. This particular configuration is based upon the driver's default configuration but contains adjusted controller gains for stiffer joint resistance. To load this file upon first use, the user selects Amplifier→Basic Setup→Load ccx File. Changes to the ccx file are saved using the floppy disk icon at the top of the GUI window. The ccx File is saved to the driver's internal flash memory using the blue integrated circuit icon near the top center of the GUI window.

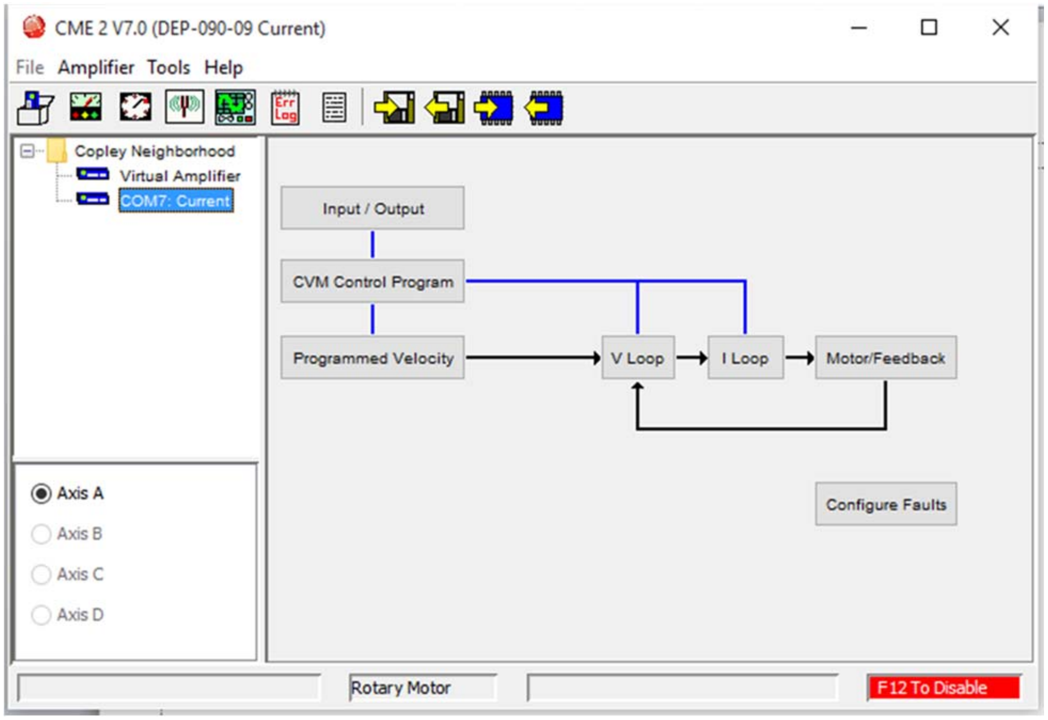
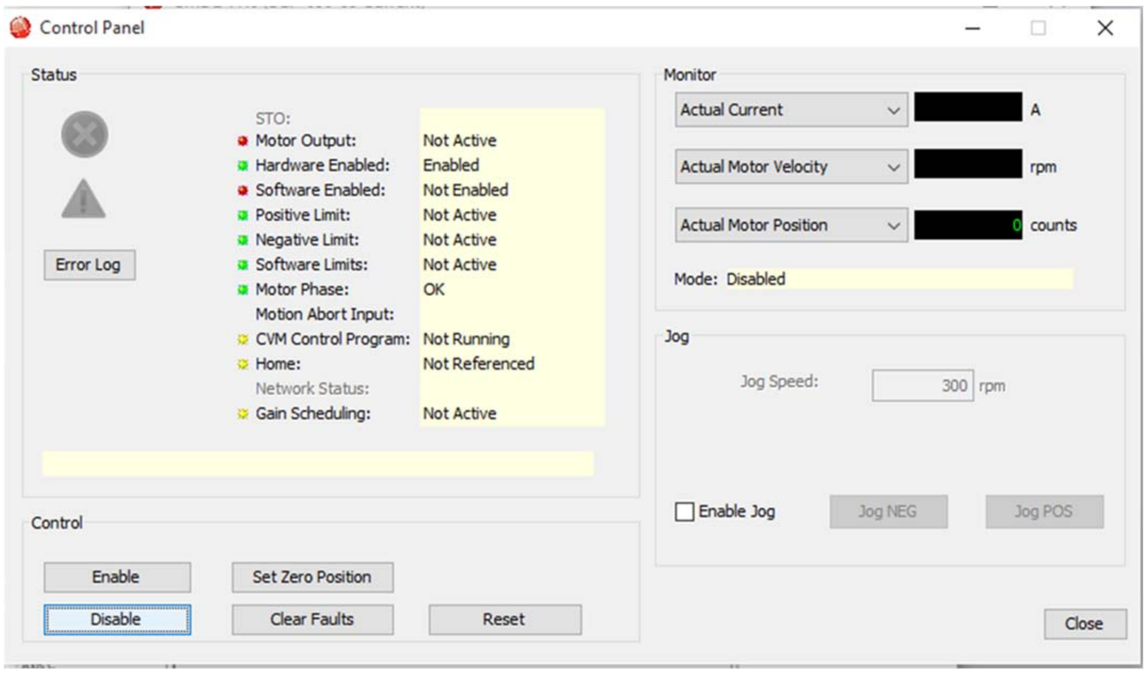


Figure 45. Copley Motion Explorer 2 (CME2) Graphic User Interface upon Initialization.

Most system troubleshooting was done within the Control Panel window, which is accessed by clicking on the second icon from the left in the main GUI's ribbon bar. To set the motor counter's zero position, the motor first needs to be disabled. Disabling the motor removes the applied voltage, which allows the user to turn the joint assembly by hand (Figure 46). This configuration takes the zero position to be when the link is aligned with the joint, or in other words, the link rotates about the joint from approximately $-\pi/2$ to $+\pi/2$.


Control Panel

Status

STO:	
Motor Output:	Not Active
Hardware Enabled:	Enabled
Software Enabled:	Not Enabled
Positive Limit:	Not Active
Negative Limit:	Not Active
Software Limits:	Not Active
Motor Phase:	OK
Motion Abort Input:	
CVM Control Program:	Not Running
Home:	Not Referenced
Network Status:	
Gain Scheduling:	Not Active

Monitor

Actual Current A

Actual Motor Velocity rpm

Actual Motor Position 0 counts

Mode: Disabled

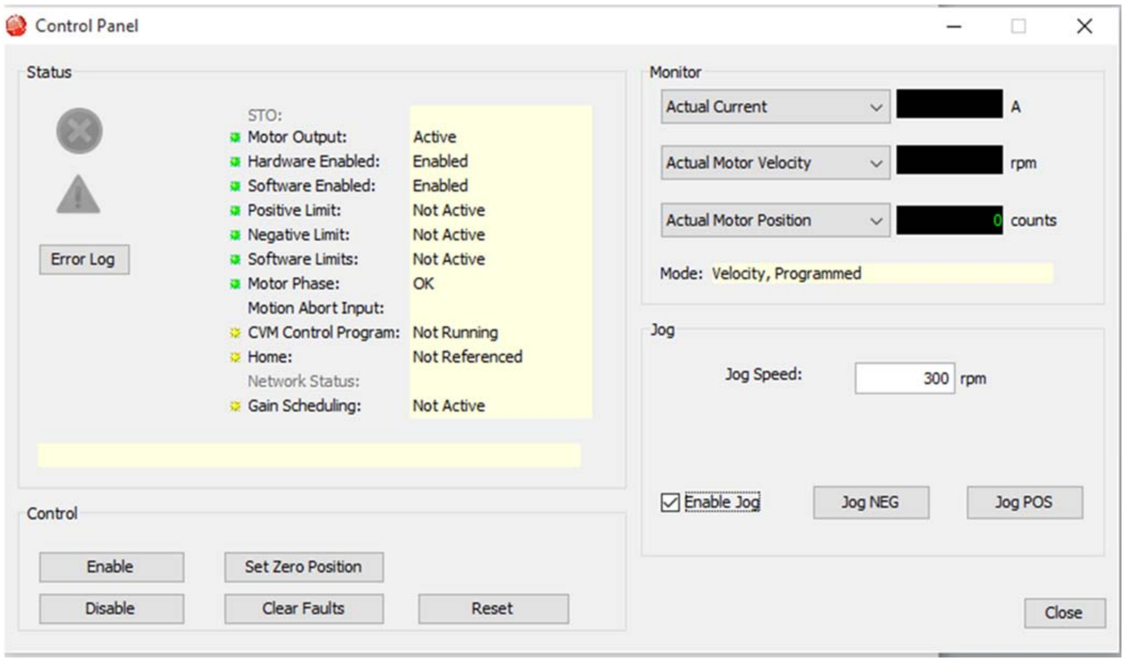
Jog

Jog Speed: 300 rpm

Figure 46. CME2 Control Panel with Disabled Motor Control.

Faults are most likely to occur if the motor is commanded to a position that it is not physically able to reach (i.e., it thinks it needs to rotate further, but the wall of the link structure is blocking the joint from turning). To prevent current overload, the controller will implement its safety mechanism (a “Latched Fault”), which will prevent the hardware from being damaged. If this happens, the motor will be unresponsive, and the AMP light on top of the driver will blink red. When a link is attached to the CME2 software, the “Clear Faults” button will reset the driver. During experimentation it is cumbersome to reattach the driver to the CME2 software, so removing and reinserting the main battery will hard boot the driver. The removal of the main battery allows for manual repositioning of the joint.

With faults cleared, the software can be enabled by clicking the “Enable” button. All indicator “lights” in the Control Panel window will show green (Figure 47). Clicking the “Enable Jog” box allows the user to send impulse commands to the motor in the positive or negative direction to ensure that the motor is responding to the driver commands.


Control Panel

Status

STO:	
Motor Output:	Active
Hardware Enabled:	Enabled
Software Enabled:	Enabled
Positive Limit:	Not Active
Negative Limit:	Not Active
Software Limits:	Not Active
Motor Phase:	OK
Motion Abort Input:	
CVM Control Program:	Not Running
Home:	Not Referenced
Network Status:	
Gain Scheduling:	Not Active

Monitor

Actual Current A

Actual Motor Velocity rpm

Actual Motor Position 0 counts

Mode: Velocity, Programmed

Jog

Jog Speed: 300 rpm

☑ Enable Jog

Figure 47. CME2 Control Panel, Faults Cleared, Jog Enabled.

With the battery connector, driver, and DC/DC converter mounted, only the mounting of the Arduino assembly remains. The Arduino assembly is mounted opposite the battery and in the current configuration, provides just enough clearance for the battery to seat snugly into the battery connector. The power and ground wires run from the battery connector to the bottom of the DC/DC converter. The power and ground wires to the Arduino assembly attach directly to this connection (nominally 14.4V). As mentioned in the hardware discussion, the Arduino assembly can accept 14.4V directly, but the Li-ion battery can hold a voltage of up to 16.8V, which will destroy the Due’s on-board voltage regulator. To prevent this from happening, a separate voltage regulator was installed that down-converts the battery’s output from 14.4V to 5V. The 5V output of the regulator is wired to the “+5V” pin on the RS232 shield. A ground wire connects the “GND” pin directly to the ground of the DC/DC converter.

The load cell amplifier jumper wires are plugged into the Arduino assembly. The load cell amplifier’s “2.7-5V” pin connects to the RS232 shield’s “+3.3V” output pin. The load cell amplifier’s data (“DAT”) pin connects to the digital input #8 pin on the RS232 shield. The load cell amplifier’s clock pin (“CLK”) connects to the RS232’s shield's #9 digital input pin. A ground-to-ground connection completes the circuit. The RS232 shield is itself connected to the servomotor driver via a serial connector and cable plugged into the driver's J5 (RS232) port.

At this point in the construction, the hardware for the power and data architectures are in place. A schematic of the power architecture is shown in Figure 48, and the data architecture schematic is shown in Figure 49. The data architecture will be discussed more thoroughly in Chapter IV.

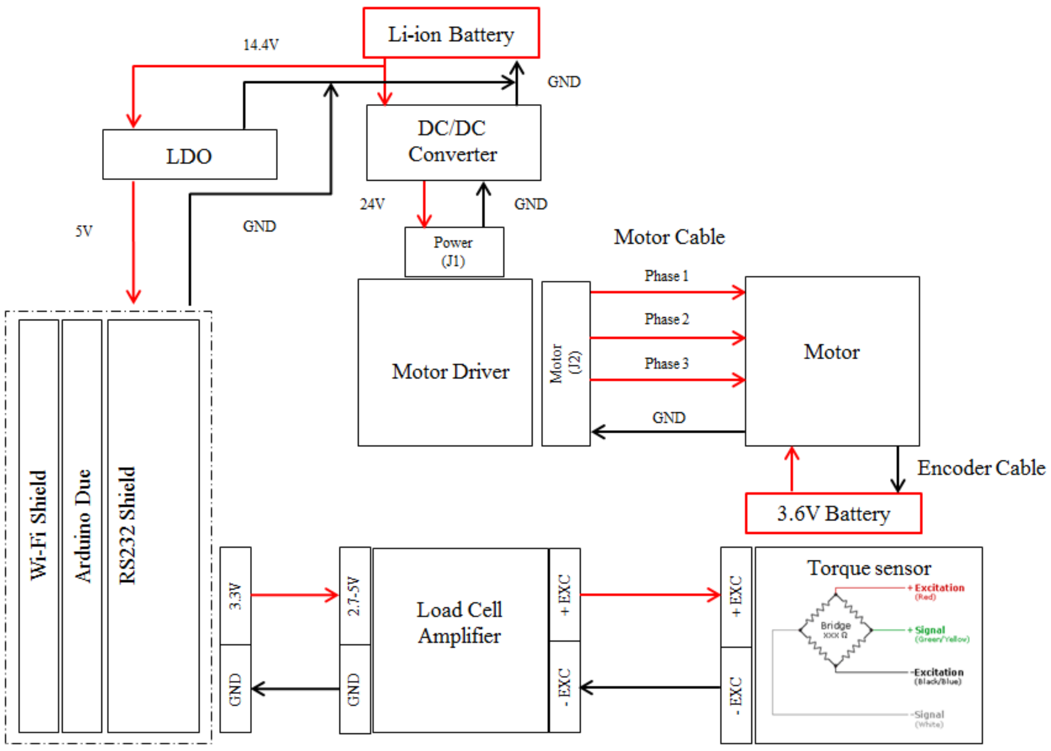


Figure 48. Power Architecture Schematic

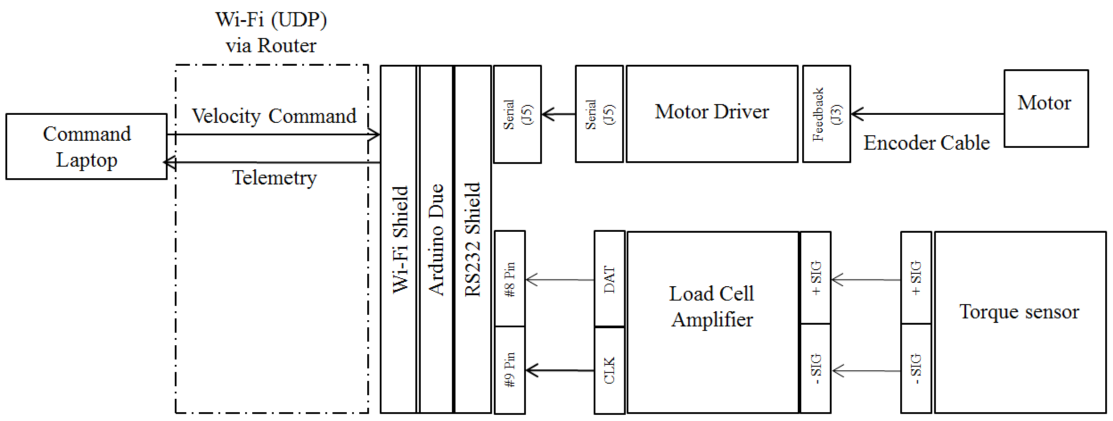


Figure 49. Data Architecture Schematic

The final step in the construction is the addition of the air bearing assembly. Three pieces of plastic tubing approximately a foot long are attached to the three ports on the plastic T-splitter. Tubing connectors are attached to each end, and the brass pipe fitting is attached to the downward-facing tube. The threaded end of the pipe fitting is wrapped in Teflon tape to ensure an air-tight seal and screwed into the bearing. The threaded ball stud is inserted first through the bearing cap before being attached to the bottom slot of the main link structure. While the ball stud may be positioned anywhere along the slot, in practice, it was positioned underneath the link center of mass. The air bearing and ball stud are saved until last because once added, the link can no longer stand on a flat surface. The H-bracket "shoe" was developed as a holder for the full link assembly. A view of the completely assembled link is shown in Figure 50.

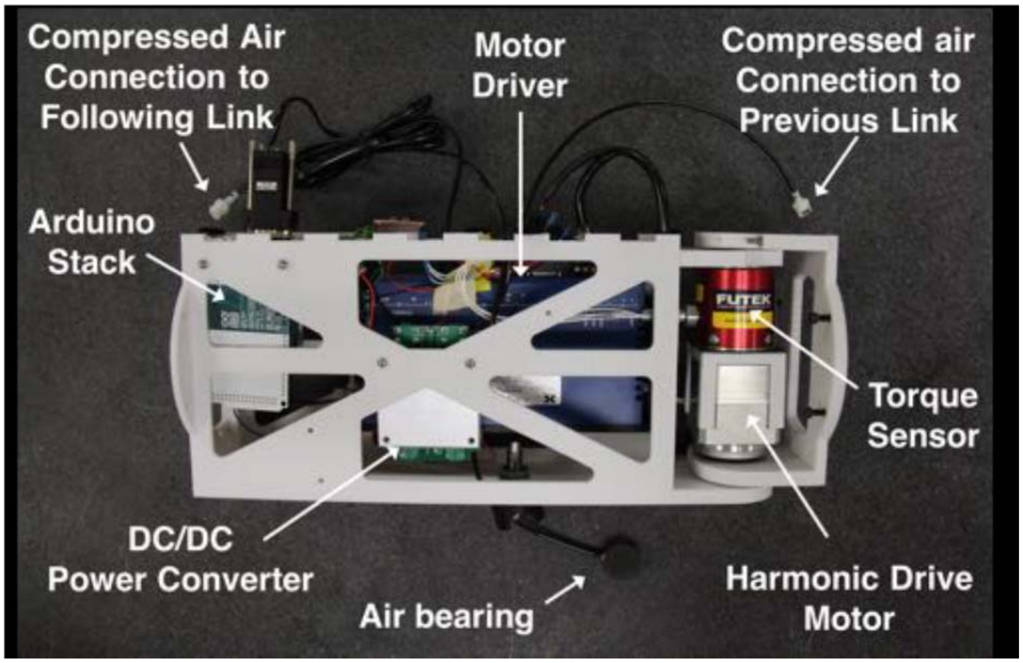


Figure 50. Complete Link Assembly. Source: NPS SRL.

With all of the hardware in place and the driver configured, a more thorough look at the link operating concept and communications architecture are necessary. An understanding of these components will enable an understanding of the microcontroller code that was implemented via the Arduino Due. The code itself proved to be the most challenging aspect of making the link work successfully. Following a discussion of the Due’s code, an explanation of the kinematics and dynamics Simulink model will be necessary before discussing the kinematic and dynamic calibration of the manipulator.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DATA ARCHITECTURE

A. OVERVIEW

This chapter explains the communications architecture of the system given its current capabilities. It outlines the concept of operations for the manipulator from a communications perspective, explains the roles of the on-board hardware and software, and discusses alternative methods of commanding the links. Finally, options for architecture improvements are discussed.

B. CONCEPT OF OPERATIONS

As in previous testing with one link, testing the four-link manipulator involved sending commands to the individual servomotor drivers from a laptop computer to the links' on-board communications hardware. Although the capability of the FSS to autonomously control each link was later demonstrated, that capability was not used for the calibration process. VICON cameras tracked the motion of the system via reflective spheres attached to the base and the outermost link. Experimentation with the four-link manipulator proved the concept that each of the links could be commanded independently. The approach is conceptually simple (as shown in Figure 51), but proved challenging in its implementation.

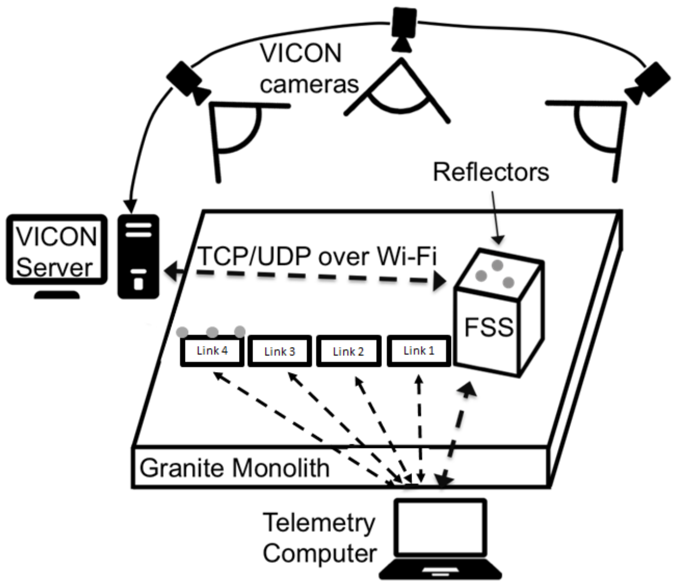


Figure 51. Diagram of the Experimental Setup. Adapted from [42].

Because neither the Arduino Wi-Fi shield nor the FSS's onboard Wi-Fi hardware are capable of creating an *ad hoc* network, the communications architecture in its current configuration depends upon an external D-Link® router. The router serves as the hub for all information flow both to and from the manipulator links. With the current hardware, two methods of sending data to and receiving data from the links are possible. In the first method, which was used for the kinematic and dynamic calibration of the manipulator, commands are sent from a laptop to each of the links, and the data from each link is received by the laptop; the FSS neither sends nor receives data. In the second method, the commanding software is compiled and uploaded to the FSS's onboard computer, and the FSS itself receives the telemetry and commands the links. Since the laptop control was the only method used in this investigation, it is the only one that will concern us here.

C. DATA FLOW

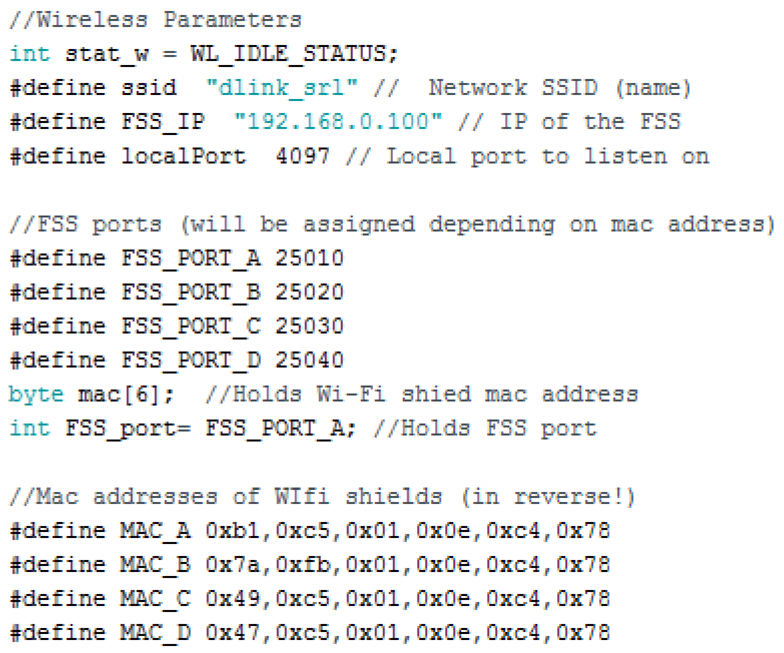
1. The D-Link Interface

A discussion of the data flow necessarily begins with the router and its web-based interface. First, when the D-Link is powered on, it creates its own network. The network is accessed by connecting to the in-house Wi-Fi network named “srl_dlink.” The interface is accessed through a web browser at the following URL: http://192.168.0.1/.

Lesson Learned

Considerable confusion during experimentation resulted because the default network name and password for the router had never been changed. D-link routers are a fairly common piece of hardware, and there was another one in the building to which the SRL hardware was connecting The router’s name was reconfigured to “srl_dlink,” and the password remained blank for ease of access.

Through the Due code, which is written in the C programming language, the Wi-Fi shields on the links are instructed to join the SRL network (Figure 52). The router employs a Dynamic Host Configuration Protocol (DHCP). That is, when devices join the network, the router assigns each device an internet protocol (IP) address, which, in the default configuration, changes each time the device joins the network. To avoid manually changing the device IPs each time in both the Simulink model and the Due code, the router was configured to assign the same IPs to the same hardware, represented by a unique media access control (MAC) address, each time. Similarly, port numbers were hard-coded into the Due code to standardize the connection process. Port numbers are not necessary for the connection itself, but they are necessary to designate the destination of the datagram packet. For example, Link 1 connects to the FSS/Control Laptop and sends data/listens to port #25010; the MAC address of its Wi-Fi shield is 0x78,0xc4,0x0e,0x01,0xc5,0xb1. The excerpt of the Arduino code below defines the network, the FSS IP and its listening port, the FSS ports, and the MAC addresses in reverse.



```
//Wireless Parameters
int stat_w = WL_IDLE_STATUS;
#define ssid  "dlink_srl" //  Network SSID (name)
#define FSS_IP  "192.168.0.100" // IP of the FSS
#define localPort  4097 // Local port to listen on

//FSS ports (will be assigned depending on mac address)
#define FSS_PORT_A 25010
#define FSS_PORT_B 25020
#define FSS_PORT_C 25030
#define FSS_PORT_D 25040
byte mac[6];  //Holds Wi-Fi shied mac address
int FSS_port= FSS_PORT_A; //Holds FSS port

//Mac addresses of WIfi shields (in reverse!)
#define MAC_A 0xb1,0xc5,0x01,0x0e,0xc4,0x78
#define MAC_B 0x7a,0xfb,0x01,0x0e,0xc4,0x78
#define MAC_C 0x49,0xc5,0x01,0x0e,0xc4,0x78
#define MAC_D 0x47,0xc5,0x01,0x0e,0xc4,0x78
```

Figure 52. Code Defining Network Parameters, Port Numbers, and MAC Addresses.

A second aspect of the data transfer process that bears discussion is the message protocol itself. The system employs the User Datagram Protocol (UDP). Unlike its more reliable cousin, the Transmission Control Protocol (TCP), UDP sends datagrams blindly without knowing for certain that there is a device on the other end to receive it [43]. This setup allows for rapid messaging and simplifies the requirement to translate the data packets, but message information can be lost during transmission or arrive in a different order [43]. For this investigation, simplified data translation was a highly desirable characteristic, but as will be discussed in a later portion of this chapter, the UDP protocol led to data management challenges.

2. The Driver Interface and the Arduino Due Code

The Due's software script initializes contact with the link's motor via on-board code. As a serial interface, the RS232 cable connects the two pieces of hardware and enables communication by text. The initialization code sets the maximum acceleration and deceleration rates (5000 counts/sec^2, an arbitrarily high number), sets the initial velocity (zero counts/sec), enables the amplifier, and requests position and velocity data from the driver. Upon receipt of the velocity command from the Simulink model, the Arduino stack passes the command to the driver. The American Standard Code for information Interchange (ASCII) strings used to achieve this functionality are shown in Table 1. For a comprehensive guide to the driver's ASCII interface and a list of available commands, see [25].

Table 3. ASCII Strings for Due-to-Driver Communication. Source: [25].

Function	**ASCII String**
Set Acceleration	s r0x36
Set Deceleration	s r0x37
Set Initial Velocity	s r0x2f
Enable Amplifier	s r0x24
Request Position Data	g r0x32
Request Velocity Data	g r0x18
Request Current Data	g r0x0c

The driver communicates the command to the motor and requests position and velocity data. The motor sends position and velocity data back via the encoder cable. Meanwhile, the torque sensor is read back through the load cell amplifier. All of the data (position, velocity, and torque) is read synchronously by the Due's script and routed back through the Wi-Fi shield, through the router, and to the UDP receive block in Simulink, which allows for analysis. Unit conversions take place within the Due script, an improvement to the previous system in which raw units from the driver (e.g., "counts") were converted to more meaningful measurements (e.g., radians) via a separate Simulink model before analysis. For the complete Arduino code, see Appendix C: OnBoardLink.ino.

3. The Simulink Model

a. The Send/Receive Switch

The Simulink block diagram in Figure 53 shows the simple on/off switch used to command a single link. For the kinematic and dynamic calibration, constant positive or negative velocities were sent via Wi-Fi to the particular link. The predetermined port numbers and IP addresses were used. On the receive side, position, velocity, torque data, and a fourth quantity, the checksum term (discussed subsequently), return over the network. In the composite ground station file, four such switches were used, one to command each of the four links.

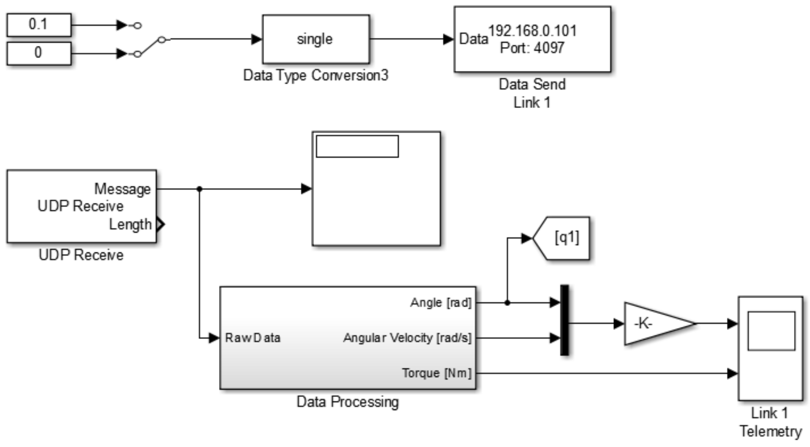


Figure 53. Send/Receive Switch for Link Control.

b. The Data Processing Subsystem

A breakout of the Data Processing Subsystem is given in Figure 54 and itself consists of two subsystems, the Checksum Subsystem and the Enabled Subsystem. The internals of the Checksum Subsystem and the Enabled Subsystem are shown in Figures 55 and 56, respectively. The Data Processing Subsystem exists to receive the data from the links, verify the integrity of the data packets, and transform those data packets into a useful form for analysis.

Before transmission from the link, each data packet initially consists of position, velocity, and torque data. Because the data stream may be corrupted during transmission, however, the microprocessor code creates a pseudo-unique checksum term based on the numerical values of the position, velocity, and torque. These four terms, then, are transmitted inside of a single data packet, which the ground station receives. This data packet passes through the Checksum Subsystem (Figure 55). The purpose of the Checksum Subsystem is to verify that a complete and uncorrupted data packet has arrived from the Link. To accomplish this, it reverses the process used to create the checksum term. The first three pieces of information (position, velocity, and torque) are stripped away from the raw data stream, multiplied by one hundred, and added together. This sum is subtracted from the original checksum term and added to one hundred. If all bits arrive uncorrupted, the final result of the additions and subtraction should be zero (to within a small tolerance).

The output of the Checksum Subsystem controls the behavior of the Enabled Subsystem. If the result of the checksum calculation is non-zero, the Enabled Subsystem is disabled, preventing the corrupt data packet from passing through. If the result of the checksum calculation is zero, the Enabled Subsystem is enabled, allowing the data packet to pass through. Since it is highly unlikely that a corrupted data packet will result in a zero calculation, the checksum ensures that only valid data packets are included for analysis.

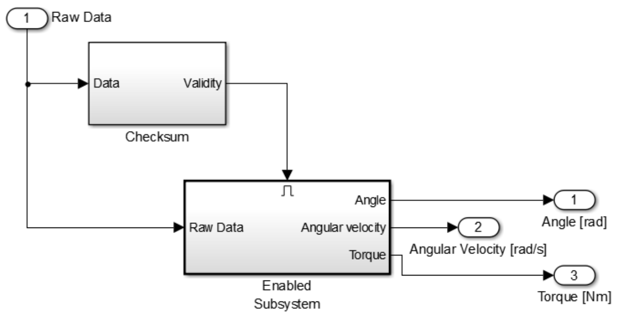


Figure 54. Breakout of the Data Processing Subsystem.

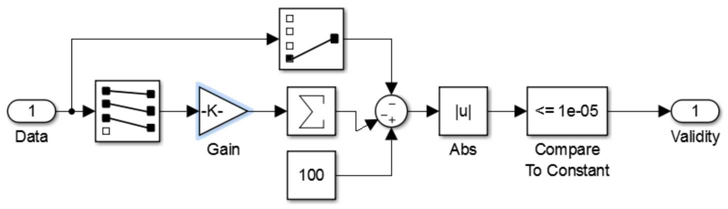


Figure 55. Breakout of the Checksum Subsystem.

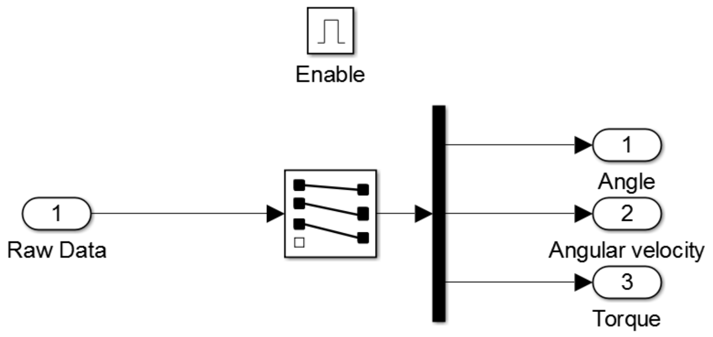


Figure 56. Breakout of the Enabled Subsystem.

The need for the checksum became apparent during initial testing of the communications architecture when data packets were dropping or being received incorrectly. The limitations of UDP were manifesting themselves. During the data transfer testing phase, it was observed that the data sent from the Arduino (position, velocity, and torque) would be received into Simulink in that order. At random intervals, however, the order of the data received would swap, usually by one place. For example, the position-velocity-torque data stream might switch to a velocity-torque-position data stream. Further, data swaps were observed that involved two variables switching places, a seemingly random behavior that made plotting and analysis of the data extremely inconvenient if not outright impossible. An extensive debugging of the Arduino's C code resulted in improved performance mainly through the manipulation of driver read delays and an increased buffer size, but the interim solution was enabling the blocking functionality in the Simulink UDP receive block.

The blocking functionality enables retention of data until a complete packet of new data is received. The maximum amount of time that Simulink will wait between packets is a configurable parameter that was arbitrarily set at ten seconds. Thus, if Simulink did not receive a new complete data packet from the Arduino subsystem within ten seconds, the simulation would time-out. Enabling the blocking mitigated the data swapping problem but introduced the new problem of the simulation timing out. In several cases, data collects of significant length (on the order of minutes) could be achieved before the simulation timed out. The interim solution was finally replaced by a better-behaved solution, the implementation of the Checksum Subsystem that detects error packets and only enables complete packets to be passed through for analysis. With the implementation of the checksum, the data swapping and system time-outs disappeared, and data could be collected with confidence.

c. The Capture Function

After any residual motion damped out of the base-manipulator system, the positions of the joint motors (q_i) as measured by the encoders were captured via Simulink's capture function, which was also included in ground station workspace (Figure 57). Depressing the capture button in Simulink enables the trigger, momentarily turning the "0" at the top of the diagram into a "1" to inform the operator that the capture has been successful. The vector of joint positions was passed into MATLAB; multiple measurements built the initial vector into a matrix containing all measurements to be used for the kinematic calibration. The Simulink capture happened at the same time as the capture of the end effector and base-spacecraft state vectors via VICON.

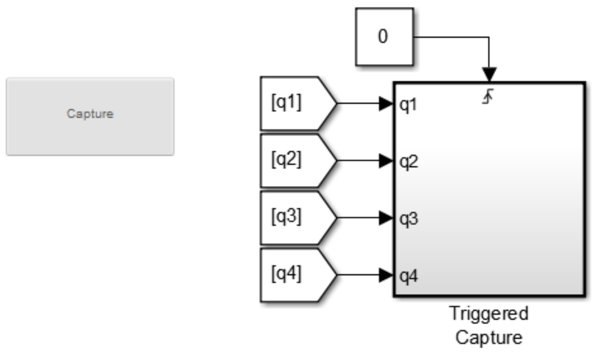


Figure 57. Capture Function

4. The VICON Interface

As it is used in the NPS SRL, the VICON system consists of ten infrared cameras, an independent server, and the proprietary software to operate the system. The motion capture cameras are depicted in the experimental setup shown in Figure 58. The array of cameras detect the light as it bounces off of a series of silver spherical markers, and this telemetry is sent to the main VICON computer via Ethernet cable where the inputs of the multiple cameras are correlated.

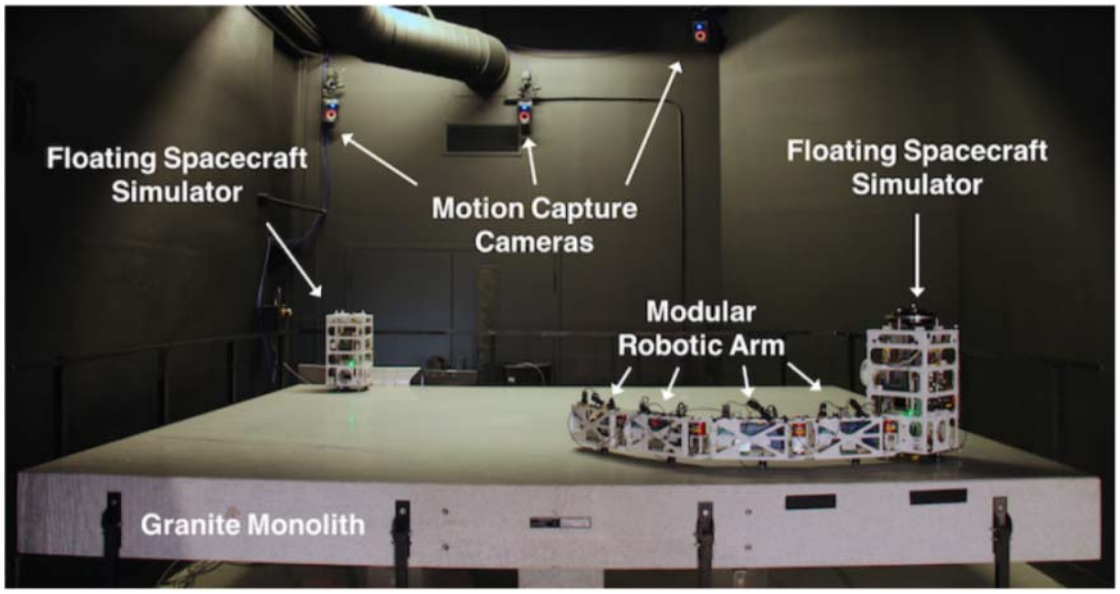


Figure 58. Experimental Setup Including VICON Cameras. Source: [44].

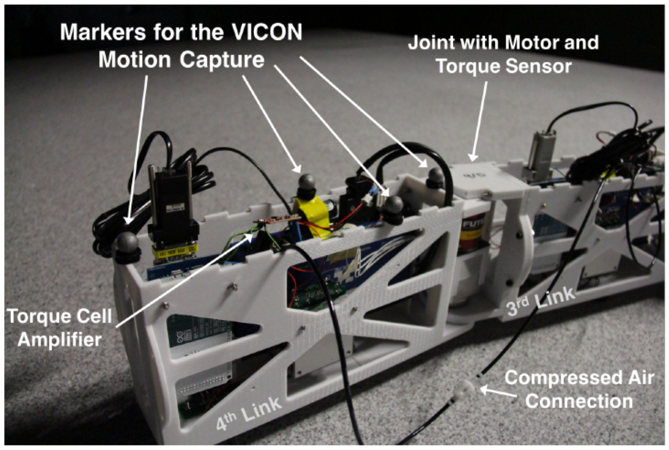

Markers for the VICON Motion Capture
Joint with Motor and Torque Sensor
Torque Cell Amplifier
3rd Link
Compressed Air Connection
4th Link

Figure 59. Silver Spheres Serve as Markers for VICON. Source: [44].

A screenshot of the VICON tracking software is shown in Figure 60 where the silver reflective spheres appear as gold balls. A reference frame is attached to the left-most ball, which is attached to the end of Link 4. This ball represents the location of the notional end effector. As with the telemetry capture from the links, the telemetry capture from VICON is accomplished manually via a capture button and collected via a MATLAB script. The interface between the VICON system and MATLAB requires VICON's proprietary Datastream SDK software to be in the MATLAB script's path. This software is available at [45]. The MATLAB script that captures the data for the kinematic calibration, including the state vectors of the base spacecraft and the end effector, is included in Appendix E: Vicon_Calibrate.m.

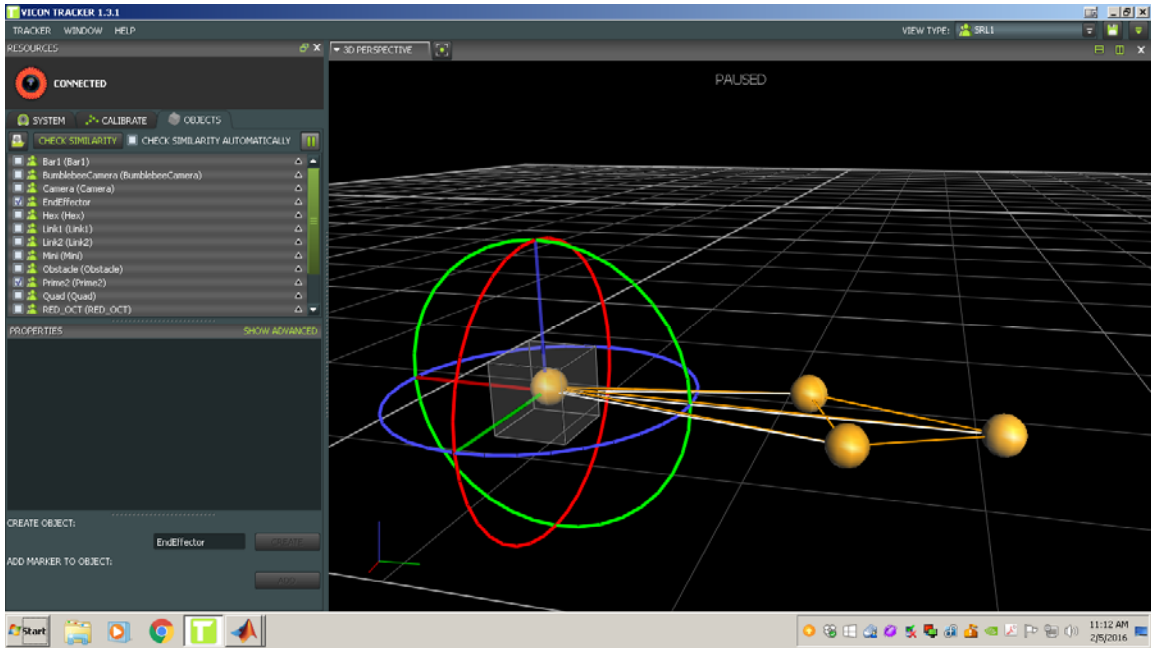


Figure 60. Screenshot of the VICON Tracking Software.

5. Potential for an *Ad Hoc* Network

Although this configuration proved adequate, its limitations were also apparent. First, the Arduino Due is not capable of creating an *ad hoc* Wi-Fi network. If it were, the microprocessor/Wi-Fi shield combination could serve as a router, and an external router would not be necessary. Such a configuration is desirable because it simplifies the hardware used in the data transmission architecture and creates a more realistic scenario; the FSS communicates with each link via a self-contained network. A more capable processor, such as a Raspberry Pi, would be capable of creating such an *ad hoc* Wi-Fi network and has the advantage of further simplifying the link hardware; a Raspberry Pi with Wi-Fi dongle can replace the whole Arduino Due, Arduino Wi-Fi shield, and the RS232 shield.

V. KINEMATICS AND DYNAMICS CALIBRATION

A. GENERAL OVERVIEW

To understand how the spacecraft/manipulator system will behave, its mechanics must be understood. The mechanics of a system consists of its kinematics and its dynamics. Broadly, kinematics is the study of the motion of an object without consideration of its causes. Dynamics is the study of the motion with consideration of its causes (i.e., forces and torques). This chapter addresses the kinematic characterization of the four-link manipulator/FSS system followed by the dynamic characterization. The mathematical underpinnings of the model are discussed in the context of their application; that is, the discussion follows the flow of the MATLAB code, the SPART model [24]. The portions of the SPART model relevant to this thesis are contained in (Appendices E–P). While the SPART model is deigned to work in for up to six degrees of freedom, the discussion of the application of the model in this section only concerns itself with the 3DOF observable in experimentation.

A second function of the SPART model is to provide the calibration functionality. The calibration is done to refine the system parameters that are difficult to measure, a process that the subsequent chapter discusses in detail. The calibration scripts, one for the kinematic calibration and one for the dynamic calibration, allow for comparison of the initially estimated system parameters and the real-world observations. With accurate knowledge of the system parameters, the system can be more precisely controlled.

B. KINEMATICS OVERVIEW

1. Definition of Terms

To characterize the kinematics of a unitary object, it is necessary to identify the position, orientation, linear velocity, and angular velocity of the object. The MARSMAN system, however, is not a unitary object, which significantly complicates the characterization of its kinematics. For the FSS with *n* link attachments, the characterization of the kinematics requires definition of the kinematic properties not only of the base, but also of each link and joint and ultimately the end effector. These parameters are highly coupled and change at each time step in the mathematical propagation.

The SPART model partly exists to model the system behavior (e.g., the positions of the links, joints, and end effector at a given time). It necessarily calculates the kinematic parameters first. This main script for kinematics is given in Appendix F: Kinematics_Serial.m. This script calls on two supporting functions. The script in Appendix G: DH_Rs.m calculate the rotation matrix R and the translation vector s based on the Denavit-Hartenberg (DH) parameters and joint variables (q_m). The script in Appendix H: Skew_Sym.m converts the designated vector into the form of a skew symmetric matrix.

2. The Base

The base itself can be treated as a rigid, unitary object with 3DOF (translation in the x and y directions and rotation about the z axis). Its position relative to the reference frame is described in terms of a state vector (q_0), a 6x1 column matrix that contains information about the base's position in terms of linear motion (x, y, z components) and rotational motion ($\theta_x, \theta_y, \theta_z$); see Equation 4. Collectively, the angles $\theta_x, \theta_y, \theta_z$ are referred to as the Euler angles.

$$q_0 = \begin{bmatrix} x \\ y \\ z \\ \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} \tag{4}$$

Since the experiments occur on a flat table, the z component of the linear motion will always be zero: the MARSMAN cannot move up and down. Similarly, it can only rotate about the z axis, so the θ_x and θ_y components can be assumed to be zero. The velocity vector ($\dot{q}_0$) and the acceleration vector ($\ddot{q}_0$) follow the same form as q_0 and are simply the first and second derivatives of q_0. As mentioned, the number of degrees of freedom of the FSS is limited by the current experimental environment, but the simulation model could easily accommodate 6DOF data if the opportunity for more-than-3DOF experiments arises.

3. The Joints

For characterizing the kinematics of an *n*-link manipulator arm, one must know the relative positions and velocities of each joint on the arm with respect to the base. The joint position vectors (q_m) and the joint velocity vector ($\dot{q}_m$) are *n*x1 column vectors. For this experimental campaign, the VICON system measured the end effector and base positions and passed them to the MATLAB/Simulink environment via Wi-Fi. The servomotor driver provided telemetry (angular position and velocity) for each link.

4. Rotation and Joint Transformation Matrices

The Euler angles of the spacecraft base are used to create the 3x3 rotation matrix (R_b) that expresses the orientation of the base with respect to the inertial frame. This model arbitrarily employs the common 1–2–3 rotation sequence shown in Equation 5 [46]. Other rotation sequences could just as readily be employed.

$$R_b = \begin{bmatrix} \cos(\theta_y)\cos(\theta_z) & \cos(\theta_x)\sin(\theta_z)+\cos(\theta_z)\sin(\theta_x)\sin(\theta_y) & \sin(\theta_x)\sin(\theta_z)-\cos(\theta_x)\cos(\theta_z)\sin(\theta_y) \\ -\cos(\theta_y)\sin(\theta_z) & \cos(\theta_x)\cos(\theta_z)-\sin(\theta_x)\sin(\theta_y)\sin(\theta_z) & \cos(\theta_z)\sin(\theta_x)+\cos(\theta_x)\sin(\theta_y)\sin(\theta_z) \\ \sin(\theta_y) & -\cos(\theta_y)\sin(\theta_x) & \cos(\theta_x)\cos(\theta_y) \end{bmatrix} \tag{5}$$

The rotation matrix only accounts for the rotational motion of the base. To account for both the rotational and translational motion, a transformation matrix is necessary. The transformation matrix of the base is of the general form shown in Equation 6 and includes the rotation matrix and a 3x1 translation vector *r* = [*x*; *y*; z] describing the base's position with respect to the inertial frame.

$$
{}^{I}T_b = \begin{bmatrix} \begin{bmatrix} & & \\ & R_b & \\ & & \end{bmatrix} & \begin{matrix} x \\ y \\ z \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \tag{6}
$$

Like the base, each of the joints also requires a transformation matrix to describe the motion of the respective joint. Whereas the transformation matrix of the base relates its position to the inertial frame, the transformation matrix of each joint relates its position to the position of the previous joint. The transformation matrices of Joints 2 through *n*+1 (where *n*+1 is the end effector) follow the same pattern. Because Joint 1 is attached to the base, its motion is described based upon its relationship to the base. Equation 7 shows Joint 1's transformation matrix, which remains constant throughout maneuvers. Joint 1 is an estimated 18.3cm from the center of mass (CoM) of the base in the x direction (the calibration process will help refine this estimate). The 3x3 identity matrix (1_3) occupying the top left of Equation 7 is indicative of Joint 1 sharing the same orientation as the base.

$$
{}^{b}T_{J1} = \begin{bmatrix} \begin{bmatrix} & & \\ & 1_3 & \\ & & \end{bmatrix} & \begin{matrix} 18.33cm \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \tag{7}
$$

To obtain the transformation matrix of Joint 1 in the inertial frame (*I*), the matrices from Equations 6 and 7 are multiplied together (Equation 8). Calculation of the transformation matrices of subsequent links requires use of DH parameters.

$$
{}^{I}T_{J1} = {}^{I}T_b \, {}^{b}T_{J1} \tag{8}
$$

5. Denavit-Hartenberg (DH) Parameters

Calculating the transformation matrices for the remaining joints (Joints 2 through *n+1*) requires recursive application of the DH parameters. The DH parameters are a convention used to conveniently define successive reference frames, each assigned to a joint of the manipulator [47]. By this method, any joint reference frame can be ultimately described in terms of the inertial reference frame. The symbolic representation of the parameters varies within literature; this thesis employs the symbols d, q, a, α. Table 2 lists the parameters and a description of their function, while Figure 61 depicts the parameters visually.

Table 4. Denavit-Hartenberg Parameters. Adapted from [44].

Parameter	Definition
$d_{i,i+1}$	Translational distance between the J_i and J_{i+1} frames along the $\hat{k}_i$ axis
$q_{i,i+1}$	Angle of rotation from $\hat{i}_i$ to $\hat{i}_{i+1}$ along $\hat{k}_i$
$a_{i,i+1}$	Distance along the common normal between $\hat{k}_i$ and $\hat{k}_{i+1}$
$\alpha_{i,i+1}$	Angle of rotation from $\hat{k}_i$ to $\hat{k}_{i+1}$ along $\hat{i}_{i+1}$

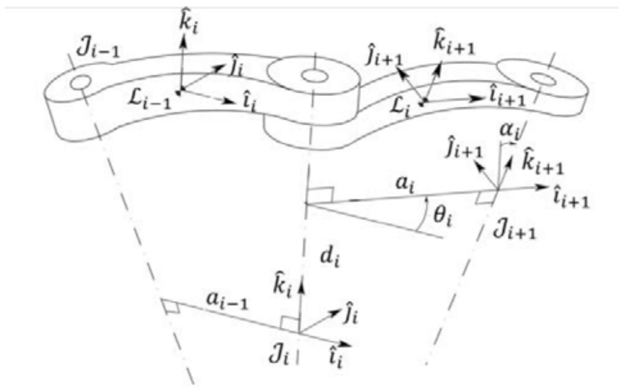


Figure 61. Visual Depiction of the Denavit-Hartenberg Parameters. Source: [44].

The generic form of the DH transformation matrix is given in Equation 9 [47]. Each joint requires its own DH transformation matrix. Since each joint is rigidly affixed to the previous link its parameters are dependent upon that link's position at any step in the numerical simulation.

$$DH = \begin{bmatrix} \cos(q_{n-1}) & -\sin(q_{n-1})\cos\alpha_{n-1} & \sin(q_{n-1})\sin\alpha_{n-1} & c_{n-1}\cos(q_{n-1}) \\ \sin(q_{n-1}) & \cos(q_{n-1})\cos\alpha_{n-1} & -\cos(q_{n-1})\sin\alpha_{n-1} & c_{n-1}\sin(q_{n-1}) \\ 0 & \sin\alpha_{n-1} & \cos\alpha_{n-1} & d_{n-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

In the case of a 3DOF, planar manipulator, the generic form shown above simplifies greatly. First, the MARSMAN is unable to translate along the *z* axis, which makes $d = 0$ throughout the simulation. Second, the *z* axes of all joints are parallel (pointing upward), so rotation about the *x* axis is never necessary when aligning reference frames. Thus, $\alpha = 0$ throughout the simulation, and the DH matrix simplifies to the form shown in Equation 10.

$$DH = \begin{bmatrix} \cos(q_{n-1}) & 1 & 0 & c_{n-1}\cos(q_{n-1}) \\ \sin(q_{n-1}) & 1 & 0 & c_{n-1}\sin(q_{n-1}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

With the DH matrix established, calculating the transformation matrices of Joints 2–4 in the inertial reference frame is only a matter of matrix multiplication. However, because the DH parameters are updated whenever a joint moves, this multiplication operation must take place numerous times. Equation 11 provides the equations for the Joint 2–4 and the end effector transformation matrices. In the kinematic calibration process, the transformation matrix of the end effector T_{EE} is the only portion that requires further manipulation.

$$\begin{aligned} T_{J2} &= T_{J1}DH_{J1} \\ T_{J3} &= T_{J2}DH_{J2} = T_{J1}DH_{J1}DH_{J2} \\ T_{J4} &= T_{J3}DH_{J3} = T_{J1}DH_{J1}DH_{J2}DH_{J3} \\ T_{EE} &= T_{J4}DH_{J4} = T_{J1}DH_{J1}DH_{J2}DH_{J3}DH_{J4} \end{aligned} \tag{11}$$

6. Link Transformation Matrices

Transformation matrices for the manipulator links are calculated much the same as the transformation matrices of the joints. The transformation matrix of Link 1 (${}^{J_{i+1}}T_{Li}$) resides in the reference frame of the *i+1* joint [23]. The vector ***b*** is the vector from the center of mass of Link *i* to the *i+1* joint and accounts for the translational motion of the link just as the vector *r* accounted for the translational motion of the spacecraft base in the base transformation matrix. Since the *i*th link rotates with the (*i+1*)th joint, the portion of the transformation matrix that describes the link's rotation is simply a 3x3 identity matrix (Equation 12). To obtain the transformation matrix of a link in the inertial frame $({}^{I}T_{Li})$, ${}^{J_{i+1}}T_{Li}$ from Equation 12 is multiplied by ${}^{I}T_{J+1}$ (Equation 13).

$$ {}^{J_{i+1}}T_{Li} = \begin{bmatrix} \begin{bmatrix} & & \\ & 1_3 & \\ & & \end{bmatrix} & \begin{matrix} -b_x^{L_i} \\ -b_y^{L_i} \\ -b_z^{L_i} \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \tag{12} $$

$$ {}^{I}T_{Li} = {}^{I}T_{J_{i+1}} {}^{J_{i+1}}T_{Li} \tag{13} $$

C. DYNAMICS OVERVIEW

At this point in the mathematical model, all of the parameters necessary for the kinematic calibration are computed, and a discussion of the property known as "twist" is necessary. Broadly, the twist vector of a link (t_i) compacts the angular and linear velocities as shown in Equation 14 [48].

$$ t_i = \begin{bmatrix} \omega_i \\ \dot{r}_i \end{bmatrix} \tag{14} $$

The model, Kinematic_Serial.m, continues by calculating the twist propagation matrices $(B_{ij}\,\text{and}\,B_{i0})$, the velocity transformation matrix (P_0), and the twist propagation vectors (p_i). These parameters are important for the characterization of the system dynamics. The twist vector, then, follows the relationship given in Equation 15 [48].

$$t_i = B_{ij} t_j + p_i \dot{q}_i \tag{15}$$

1. Twist Propagation Matrices (B_{ij} and B_{i0})

The 6x6 twist propagation matrix B_{ij} accounts for angular and linear velocities of pairs of joints by taking the positional difference $(r_j - r_i)$ of successive joints and turning that vector into a 3x3 skew symmetric matrix $r_{ji}^{\times}$ [48]. Similarly, the twist propagation matrix B_{i0} accounts for the angular and linear velocities between the i^{th} link and the base $(r_0 - r_i)$ or $r_{0i}^{\times}$. Interestingly, the skew symmetric functionality is not built into MATLAB as an organic function, so the in Appendix H: SkewSym.m was again employed to convert the r_{ji} and r_{0i} vectors into the necessary form. The form of B_{ij} is shown in Equation 16 where 1_3 denotes a 3x3 identity matrix and 0_3 denotes a 3x3 zero matrix [48]. The form of B_{i0} is the same with the exception that $r_{0i}^{\times}$ replaces $r_{ji}^{\times}$.

$$B_{ij} = \begin{bmatrix} I_3 & 0_3 \\ r_{ji}^{\times} & I_3 \end{bmatrix} \tag{16}$$

2. Velocity Transformation Matrix (P_0) and Twist Propagation Vector (p_m)

The velocity transformation matrix accounts for the twisting effects of the spacecraft base on the system dynamics. It is shown in Equation 17, "a 6x6 matrix that contains the base-spacecraft rotation matrix" in the inertial frame [48].

$$P_0 = \begin{bmatrix} {}^{I}R_{L0} & 0 \\ 0 & I_3 \end{bmatrix} \tag{17}$$

The twist propagation vectors account for the twisting effects of each joint on the system dynamics. An explanation of the twist propagation vector requires introduction of the $r_i, l_i, e_i,$ and g_i vectors. Figure 62 depicts these vectors pictorially.

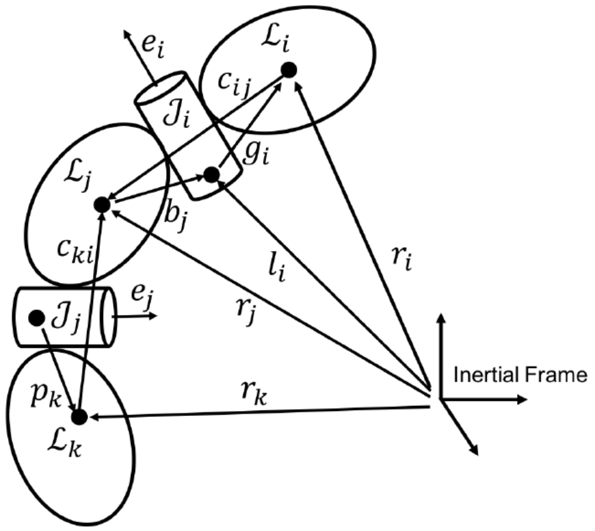


Figure 62. Depiction of Dynamic Model Vectors: Source: [48].

Fortunately, these vectors do not require new calculations; they are made by parsing pieces from already existing transformation matrices. For example, r_i is the first three terms of fourth column of the i^{th} link transformation matrix (as seen previously) and accounts for the link's translational motion in inertial space; in other words, r_i is the vector from the origin of the inertial frame to the center of mass of the i^{th} link. The vector l_i is similar to r_i but comes from the joint transformation matrix (instead of the link transformation matrix) in the inertial frame. It accounts for the joint's translational motion in inertial space and represents the vector from the origin of the inertial frame to the origin of the i^{th} joint reference frame. The vector e gives the orientation of the joint's rotation axis (the first three rows of the third column of each joint transformation matrix). In the planar case, e is simply a unit vector indicating rotation about the *z* axis. Finally, the geometric vector g_i is the difference between r_i and l_i. Since the joints in this experiment are revolute, the magnitude of g_i does not change. With these four vectors defined, the manipulator twist propagation vector (p_m), is calculated as shown in Equation 18 [48].

$$p_i = \begin{bmatrix} e_i \\ e_i \times g_i \end{bmatrix} \tag{18}$$

D. KINEMATIC CALIBRATION

The explanation thus far has dealt with the mathematical manner in which the nominal kinematics of the system are calculated largely via Kinematic_Serial.m and its supporting functions. The kinematic calibration process is primarily concerned with using the model and the experimentally measured position of the end effector to refine the system parameters. To that end, the end effector was measured in a series of poses.

With the four-link manipulator attached to the base spacecraft and floating on the monolith, the manipulator links were given a number of random commands (*m*). Following each command, the state of the end effector (x_{EE}) was measured using the VICON system, and the measurements of all joint angles (as read by the driver and passed through UDP to Simulink) were recorded in an *mxn* matrix (*q)*. When combined with the VICON measurements of the base spacecraft position (*q0*), the homogeneous transformation matrix describing the end effector position (T_{EE}) can be solved for numerically. From this, the *x* and *y* translational and the θ_z rotational components of the end effector in the inertial frame are extracted.

Based on the kinematic model's estimates of the end effector state, small variations were induced into the *x, y,* and θ_z measurements to see how much a small change in each parameter would affect the output of the kinematic model (k). These partial derivatives were collected into the kinematic calibration matrix (Φ), which is of the general form shown in Equation 19. Again, the planar case simplifies the calculations. The parameters d and α do not change over time, so they are inconsequential to the calculations. The q parameter for all four joints and the a parameters for all four links, however, must be incrementally adjusted to properly tune the joint measurements.

Further, refinement of the location of Joint 1 with respect to the base's center of mass is necessary, so the parameters $x_{B,J1}, y_{B,J1}, \theta_{B,J1}$ must be similarly manipulated.

$$\Phi = \left[\delta k / \delta a \quad \delta k / \delta d \quad \delta k / \delta x_{B,J1} \quad \delta k / \delta y_{B,J1} \quad \delta k / \delta \theta_{B,J1} \right] \tag{19}$$

With each iteration, the nominal end effector state vector (x_n) was compared to the experimentally measured end effector state vector (x_m) and the differences compiled into the matrix Δx (Equation 20) [47].

$$\Delta x = x_m - x_n \tag{20}$$

The matrix $\Delta\zeta$ is a combination of the of the kinematic calibration matrix (Φ) and the nominal/measured differences (Δx) as shown in Equation 20. An iterative batch least-squares method was used to drive down the values of $\Delta\zeta$, the "parameter variations with respect to the nominal values" [47]. When the least-squares solution returned a $\Delta\zeta$ vector within a pre-determined tolerance (Equation 21), the solution provided insight into how much the estimated parameters, varied from the true system behavior.

$$\Delta\zeta = \left(\Phi^T \Phi\right)^{-1} \Phi^T \Delta x \tag{21}$$

E. DYNAMIC CALIBRATION

1. Overview

While the kinematic calibration concerns itself with refinement of kinematic parameters, the dynamic calibration concerns itself with refinement of the dynamic parameters. The two calibration processes work hand-in-hand; the offsets determined by the kinematic calibration feed the dynamic calibration model as an initial guess. Thus, it is not possible to complete the dynamics calibration without first completing the kinematics calibration.

Functionally, both processes share similarities. The same Simulink controller, VICON system setup, and telemetry data capturing technique was used for the dynamic calibration. All measurements were taken with the spacecraft/manipulator system in a stationary position; measurements were not taken while the motors were active or while the residual vibration from the maneuver was naturally damping out. Unlike the kinematics calibration, the dynamics calibration required capturing the three sets of position data both prior to a maneuver and following the maneuver: the initial (q_{01}) and final (q_{02}) positions of the base, the initial (q_{i1}) and final positions (q_{i2}), and the initial (x_{ee1}) and final (x_{ee2}) positions of the end effector. The main script for the dynamics calibration is in Appendix K: MARSMAN_DynCal, but before proceeding with the actual calibration, several supporting functions are employed to account for the system dynamic effects and produce—as an interim objective—the nominal state vector of the base-manipulator system $(q_{0x}, q_{0y}, q_{0\theta})$.

To obtain the nominal state vector, the link and body rotation matrices (R_L and R_0, respectively; both are outputs of Kinematics_Serial.m) are first used to construct the inertia matrices of the base (I_0) and manipulator (I_m). The inertia matrices and the twist propagation matrices (B_{ij} and B_{i0}, also products of Kinematics_Serial.m) contribute to the mass composite body matrices of the base $(\tilde{M}_0)$ and the manipulator $(\tilde{M}_m)$. With these, the generalized inertia matrices of the base (H_0) and manipulator (H_m) can be calculated and used to determine the derivative of the base position $(\dot{q}_0)$. The vector is integrated and variation applied to the components $(q_{0x}, q_{0y}, q_{0\theta})$ for construction of the dynamic calibration matrix (Φ).

2. Inertia Matrices

The link and body rotation matrices (R_L and R_0) from Kinematics_Serial.m are passed to a supporting function (see Appendix L: I_I.m). The body rotation matrix (R_0) is multiplied by the previously defined inertia of the spacecraft to produce the inertia matrix of the base in the inertial frame (I_0). A series of *n* inertia matrices in the inertial frame (one for each link) are calculated based on Equation 22.

$$I_{mi} = {}^{I}R_{Li} I_i^{Li} \, {}^{I}R_{Li}^{T} \tag{22}$$

3. Mass Matrices

The link mass matrices (M_i) and the base mass matrix (M_0) takes the general form shown in Equation 23 and takes into account the inertia (*I*) and mass (*m*) properties of each link. In the figure below, $1_{3,3}$ represents a 3x3 identity matrix [48].

$$M = \begin{bmatrix} I & 0 \\ 0 & m1_{3,3} \end{bmatrix} \tag{23}$$

From a physical perspective, the mass and inertia properties of the links and of the base effect the twisting of the joints. The mass composite body matrices of the links $(\tilde{M}_m)$ take into account these effects (encapsulated in the twist propagation matrices) by first computing the effects at the n^{th} joint and conducting a backwards recursion. When the recursion has accounted for all links, $\tilde{M}_m$ is the end result (Equation 24). The mass composite body matrix of the base $(\tilde{M}_0)$ is computed using the B_{i0} twist propagation matrix and the mass composite body matrix produced by the recursion (Equation 25) [48]. The code for the function used to conduct these calculations is given in Appendix M: MCB_Serial.m.

$$\tilde{M}_m = M_{m,i} + B_{ij,i}^T \tilde{M}_{m,i+1} B_{ij,i} \tag{24}$$

$$\tilde{M}_0 = M_0 + B_{i0}^T \tilde{M}_{m1} B_{i0} \tag{25}$$

4. Generalized Inertial Matrices

With $\tilde{M}_0, \tilde{M}_m, B_{ij}, B_{i0}, P_0, p_m$, the generalized inertia matrix of the base (H_0) and the generalized coupling inertia matrix of the manipulator and the base (H_{0m}) can be computed. The script that accomplishes this calculation is given in Appendix N: GIM_Serial.m. The calculation of H_0 follows the matrix multiplication given in Equation 26 [48]. Similarly, the coupling inertia matrix, which accounts for the twists of the joints, the inertias and masses of the links, and how these affect the inertia of the base-manipulator system is given in Equation 27 [48].

$$H_0 = P_0^T \tilde{M}_0 P_0 \quad (26)$$

$$H_{0m} = p_m^T \tilde{M}_{m,i} B_{i0,\mathrm{i}} P_0 \quad (27)$$

5. Determining the Base-Manipulator System State (q)

The goal of calculating inertias, mass composite body matrices, and generalized inertia matrices is to determine the base-manipulator's state (q), which consists of the base state vector and the manipulator state vector as shown in Equation 28.

$$q = \begin{bmatrix} q_0 \\ q_m \end{bmatrix} \quad (28)$$

The time derivative of the base's state vector $(\dot{q}_0)$ is calculated by Equation 29, which is based on the conservation of momentum principle and includes previously calculated inertia matrices [48]. The manipulator joint velocities $(\dot{q}_m)$ are calculated by subtracting the initial measured position of each joint from the final measured position of each joint and dividing by the time of the maneuver (Equation 30). The calculation of $\dot{q}$ is accomplished by the function in Appendix O: q_dot_fun.m.

Since the state of the base is only a function of the manipulator path, the experimentation can exploit a non-holonomic constraint. That is, since only one joint is moved at a time, the time over which the movement occurs is irrelevant; the system will achieve the same position whether the maneuver takes one or one hundred seconds. For the purposes of the MATLAB calculations, all maneuver times (Δt) were set to be five seconds. The derivative is simply integrated to obtain the state vector of the system (q) and the components of $(q_{0x}, q_{0y}, q_{0\theta})$ are used in the calibration proper. See Appendix P: q0_maneuver.m for the integration function.

$$\dot{q}_0 = -H_0^{-1} H_{0m} \dot{q}_m^T \tag{29}$$

$$\dot{q}_m = (q_{m,i2} - q_{m,i1}) / \Delta t \tag{30}$$

6. Inducing Variation, the Dynamic Calibration Proper

As in the kinematics calibration, in the dynamics calibration, small variations are applied to the parameters of interest—in this case, the ***b*** vector and the inertia *I.* These small variations are passed through the dynamics model. These partial differentials are used to form the dynamics calibration matrix, which, like the kinematics calibration matrix is denoted as Φ. In this case however, Φ depends upon how much the outputs of the dynamics model (d) change with changes in the parameters of interest (Equation 31).

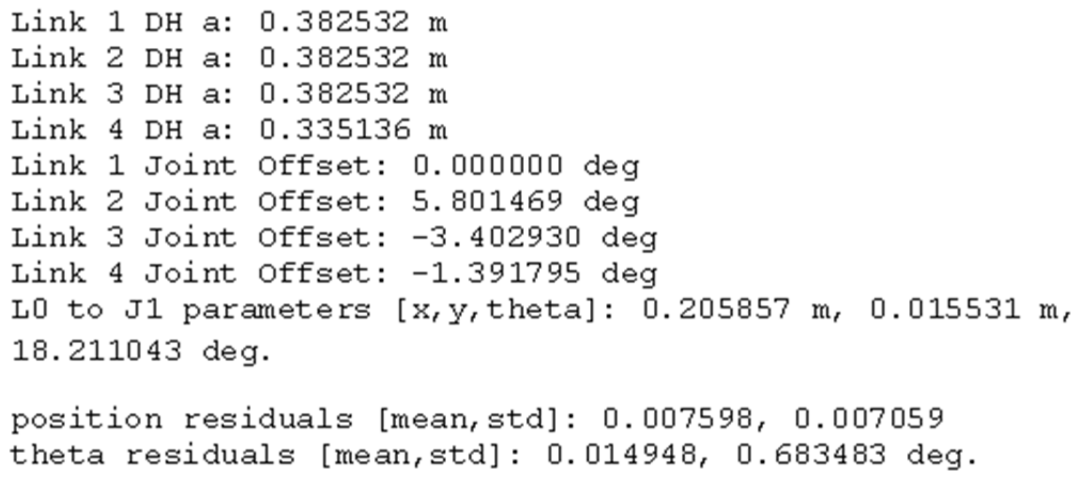
$$\Phi = \left[\delta d / \delta b \quad \delta d / \delta I \right] \tag{31}$$

Figure 63. Dynamic Calibration Matrix.

The dynamics model (see Appendix P: q0_maneuver.m) integrates the joint positions to ultimately predict the coordinates of the system state vector in the inertial frame. As in the kinematics calibration, Φ and the matrix of nominal/measured differences (Δx) are used to construct $\Delta\zeta$, and a least-squares method is used to refine the parameters. Ultimately, the dynamics calibration provided insight into how much the link inertias (I_i) and the ***b*** vectors varied between theoretical prediction and experimental measurement.

F. ANALYSIS

Achieving a valid least-squares solution to the kinematics and then the dynamics calibration defined the geometry and mass properties of the base-manipulator system. In the planar case with revolute joints, DH parameter a for the links was calculated to be 0.382532 m for Links 1–3 and slightly shorter for Link 4. This conclusion makes sense because Link 4 is slightly shorter than the other links due to the lack of an attached joint or end effector (Figure 64). Also, while a deliberate effort was made to initially position all joints at a zero angle, the calibration process showed that the joints were actually slightly offset. Note that Link 1's joint offset is 0.00 degrees, while the offset of the base (L0) is 18.21 degrees. Since Joint 1 is attached to the base, it could equivalently be said that the offset of the base is 0.00 degrees while the offset of Joint 1 is 18.21 degrees. Overall, the mean position residual of the system was thousandths of a meter, and the mean angular residual was on the order of hundredths of a degree—indicators that the kinematic calibration was successful.



```
Link 1 DH a: 0.382532 m
Link 2 DH a: 0.382532 m
Link 3 DH a: 0.382532 m
Link 4 DH a: 0.335136 m
Link 1 Joint Offset: 0.000000 deg
Link 2 Joint Offset: 5.801469 deg
Link 3 Joint Offset: -3.402930 deg
Link 4 Joint Offset: -1.391795 deg
L0 to J1 parameters [x,y,theta]: 0.205857 m, 0.015531 m,
18.211043 deg.

position residuals [mean,std]: 0.007598, 0.007059
theta residuals [mean,std]: 0.014948, 0.683483 deg.
```

Figure 64. MATLAB Results of the Kinematics Calibration.

The dynamics calibration did not provide as precise a solution as the kinematics calibration. The inertias of the links were all calculated to be the same, and the *b* vectors, with the exception of the shorter Link 4 were also determined to be the same (Figure 65). The mean and standard deviation of the data set, however, were considerably larger than seen in the kinematics portion. This decrease in precision is partially due to the almost imperceptible motion of the base due to Link 4's motion, and sensitivity of the system to residual motion after the cessation of movement commands. During experimentation, it was noticed that the system tended to drift after it should have come to a stable stop, possibly due to a misaligned air-pad, a piece of air tubing dragging on the surface of the table, or slight imperfections on the surface of the monolith.

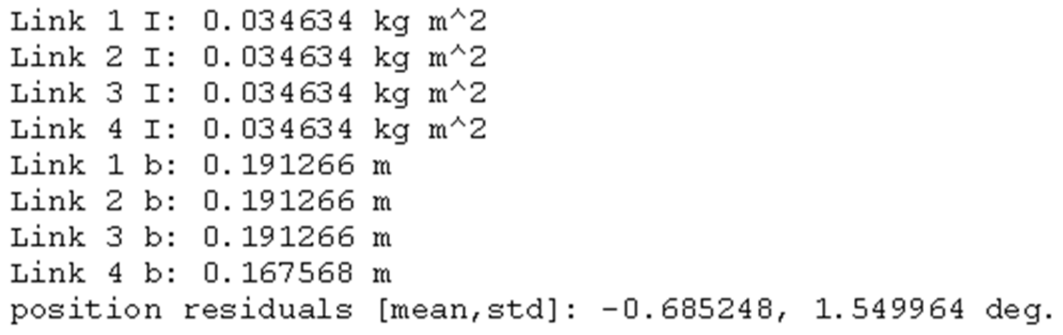

```
Link 1 I: 0.034634 kg m^2
Link 2 I: 0.034634 kg m^2
Link 3 I: 0.034634 kg m^2
Link 4 I: 0.034634 kg m^2
Link 1 b: 0.191266 m
Link 2 b: 0.191266 m
Link 3 b: 0.191266 m
Link 4 b: 0.167568 m
position residuals [mean,std]: -0.685248, 1.549964 deg.
```

Figure 65. MATLAB Outputs of the Dynamic Calibration.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

A. SUMMARY OF WORK

This thesis built upon several years of work at the Naval Postgraduate School Spacecraft Robotics Laboratory and took advantage of an established laboratory environment with a state-of-the-art motion tracking system, granite monolith, and already-constructed Floating Spacecraft Simulator (FSS) platform. The initial link design was improved by adopting an all commercial-off-the-shelf hardware components, changing the communications architecture to one that uses Wi-Fi, and coding more effective software for the processing of the system's data. The initial six months of the effort centered around gaining an understanding of the previous work and the integration of the new hardware components and software. Glitches in software development caused significant delay but were ultimately overcome.

With the second link completed, the design improvements were retrofitted onto the original link, and the third and fourth links were constructed in relatively little time. With the four-link configuration constructed and attached to the FSS, the kinematic and dynamic calibration processes began. MATLAB and Simulink tools, which became the Spacecraft Robotics Toolkit (SPART), were developed and validated via table-top experimentation. The work for this thesis contributed to three conference papers. Two were presented at the 6th International Conference on Astrodynamics Tools and Techniques (ICATT), and an abstract for the third has been accepted for the American Institute of Aeronautics and Astronautics' (AIAA) Space and Aeronautics Forum and Exposition in September 2016 [23], [42], [49]. Additionally, a patent application is being pursued.

B. FUTURE WORK

Due to the modular nature of the manipulator links, future research can take a number of different directions. In the near term, however, work will likely continue with the four-link manipulator configuration.

(1) How Small is Small?

Throughout the literature on the subject, the term “small spacecraft” is used when describing a spacecraft whose motion is significantly dynamically coupled with the motion of the manipulator. As discussed in the introduction, the ratio of the base mass/inertia to the manipulator mass/inertia are useful measures of how heavily the dynamics are likely to couple. There is, however, no apparent effort to quantify at what ratios the dynamics become “significantly” coupled. How big does the manipulator have to be to affect the base dynamics? How small does the manipulator have to be for its effects to be ignored? With the tools developed in the SRL, this topic could be investigated via simulation and table-top experimentation.

(2) Explore Manipulator Control Algorithms

The intent of the kinematics and dynamics calibration (and all the proceeding hardware and software development) was to put in place the pieces for a controllable manipulator. As a stepping stone, efforts to control the manipulator not from a command laptop (as done in the experimentation for this thesis) but from the FSS’s onboard computer are underway. Other areas ripe for investigation include the most effective manner in which to employ a manipulator with respect to time or power usage; these are two areas in which optimal control theory could be explored.

The current investigation focused on a four-link serial manipulator, but the modularity of MARSMAN allows for exploration of different topologies. For example, with the in-house additive manufacturing capability, a serial manipulator that branches into multiple arms could be constructed. Further, closed-tree topologies such as two two-link manipulators working in concert as a pincer could be investigated. These topologies are significantly more difficult to analyze and require the employment of differential algebraic equations (DAE). Whatever the unique configuration, different configurations of the modular links will require adaptations of the kinematics and dynamics calibration and present their own control challenges

(3) Construct and Integrate an End Effector

This project focused on the construction of the manipulator links themselves, and the mathematical modeling assumed a notional end effector was attached. An end effector is necessary for proximity operations that aim to capture a target FSS. A simple end effector could consist of a Velcro surface mounted to both the target vehicle and the end of the manipulator. More complex end effectors, like grippers, provide additional opportunities for hardware construction and software integration. Yale University's OpenHand Project provides computer aided drafting (CAD) files for a variety of manipulators that can be assembled via additive manufacturing. The tendons for the gripper are constructed by pouring resin into a 3-D printed mold [50]. With a gripper, rendezvous and capture problems can become an active area of research.

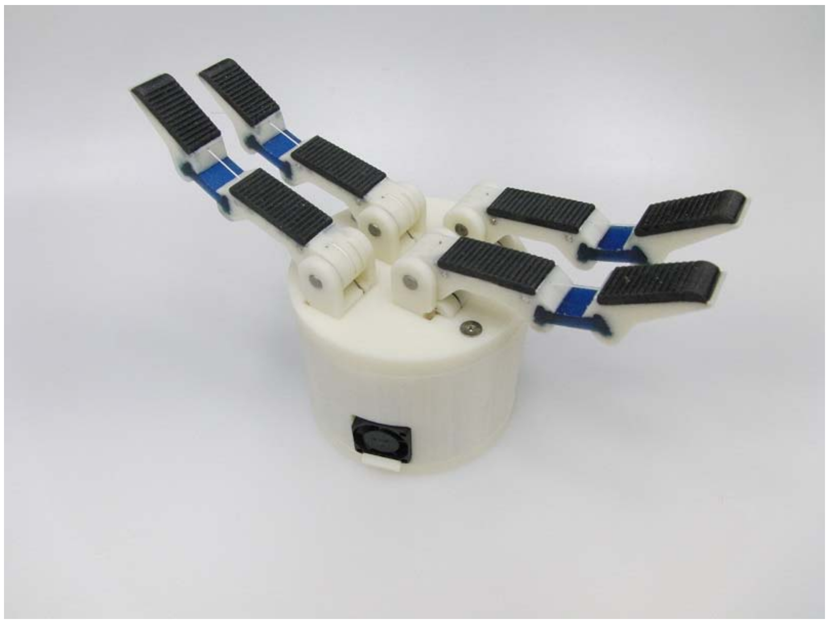


Figure 66. Sample End Effector from the Yale OpenHand Project. Source: [50].

(4) Rendezvous and Capture Operations

With viable control algorithms and an end effector, rendezvous and capture operations become possible. Initial attempts may consider capturing a stationary spacecraft with the eventual goal of de-tumbling a spinning FSS, perhaps while maintaining visual observation via a camera mounted on the base spacecraft. This line of research is synergistic with the current research efforts at the SRL for FSS-to-FSS docking. From a guidance, navigation, and control (GNC) perspective, this problem becomes particularly challenging because it requires not only control of the manipulator itself but also control of the base spacecraft via its thrusters and reaction wheels.

(5) Hardware Upgrades

The Arduino Wi-Fi shield is no longer in production. As components fail, they will need to be replaced with a comparable capability. The Raspberry Pi microcontroller has the potential to replace the Arduino Wi-Fi shield, the Arduino Due, and the RS232 shield because it can serve as microcontroller, create its own *ad hoc* Wi-Fi network, and provide the necessary connection ports to the motor driver. The creation of the *ad hoc* Wi-Fi network will allow for the links, the FSS, and a telemetry laptop to communicate directly in a more realistic and reliable manner. With this capability, the D-link router can be removed from the communications architecture. Further, the Raspberry Pi allows for a simplified interface to control the end effector and the possibility of attaching a small camera for use in proximity operations.

(6) The Self-Assembling Manipulator

Apart from organic propulsion, the current design of the manipulator links is such that they are very nearly miniature FSSs. With modification to the current modular design, each link could be made to operate independently, and the links themselves could rendezvous to form a workable manipulator that could then be used to perform a useful task.

(7) A Real Manipulator Spacecraft

The current base-manipulator system is a test-bed configuration never intended to fly in space. While the development of a flight-worthy base-manipulator system is outside the current scope of the SRL's research efforts, the development of such a concept could easily provide research material for students in structures, thermal systems, power systems, and GNC. The development of such a spacecraft would be a worthy of exploration during NPS's capstone design sequence.

C. RESEARCH SIGNIFICANCE

The research conducted for this thesis is significant in several ways. As mentioned previously, the initial development of a modular robotic manipulator link in 2014 was the first of its kind. To the knowledge of the author, the subsequent hardware and software improvements and the construction of the four-link manipulator from modular links is unique in academia. The kinematic and dynamic models, elucidated by SRL writings such as [48], was implemented via MATLAB and, along with the calibration code, made publicly available to anyone wishing to investigate spacecraft with attached manipulators—another first-of-its-kind effort to share knowledge with the larger space robotics community. The research work over the past year has contributed to two published conference papers; a third conference paper abstract based on the work has been accepted for the fall of 2016. Finally, the manipulator is being used to teach graduate students robotics and multi-body mechanics at NPS.

Ultimately, this project implemented the highly non-linear dynamical model of a small spacecraft with an attached robotic manipulator consisting of multiple modular links. The concept was validated through a table-top experimentation campaign that proved the concept of controlling multiple links independently to complete the kinematic and dynamic calibrations. The experimental framework will allow further exploration into spacecraft/robotic mechanics, control theory, and operational concepts ranging from docking and capture maneuvers to spacecraft servicing. The beauty of the MARSMAN system is that a new piece of hardware does not need to be funded and built each time a new line of inquiry appears. The modular links can be reconfigured for any number of mission scenarios that may be of interest to civil, military, or commercial space entities.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SPARKFUN HX711 LOAD CELL AMPLIFIER OPEN-SOURCE HEADER FILE [51]

```
#ifndef HX711_h
#define HX711_h

#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

class HX711
{
     private:
          byte PD_SCK;   //  Power  Down  and  Serial  Clock
Input Pin
          byte DOUT;          // Serial Data Output Pin
          byte GAIN;          // amplification factor
          long OFFSET;   // used for tare weight
          float SCALE;   // used to return weight in grams,
kg, ounces, whatever

     public:
          // define clock and data pin, channel, and gain
factor
          //  channel  selection  is  made  by  passing  the
appropriate gain: 128 or 64 for channel A, 32 for channel B
          // gain: 128 or 64 for channel A; channel B works
with 32 gain factor only
          HX711(byte dout, byte pd_sck, byte gain = 128);

          virtual ~HX711();

          // check if HX711 is ready
          //  from  the  datasheet: When  output  data  is  not
ready  for  retrieval,  digital  output  pin  DOUT  is  high.
Serial clock
          // input PD_SCK should be low. When DOUT goes to
low, it indicates data is ready for retrieval.
          bool is_ready();

          // set the gain factor; takes effect only after a
call to read()
```

```
          // channel A can be set for a 128 or 64 gain;
channel B has a fixed 32 gain
          // depending on the parameter, the channel is
also set to either A or B
          void set_gain(byte gain = 128);

          // waits for the chip to be ready and returns a
reading
          long read();

          // returns an average reading; times = how many
times to read
          long read_average(byte times = 10);

          // returns (read_average() - OFFSET), that is the
current value without the tare weight; times = how many
readings to do
          double get_value(byte times = 1);

          // returns get_value() divided by SCALE, that is
the raw value divided by a value obtained via calibration
          // times = how many readings to do
          float get_units(byte times = 1);

          // set the OFFSET value for tare weight; times =
how many times to read the tare value
          void tare(byte times = 10);

          // set the SCALE value; this value is used to
convert the raw data to "human readable" data (measure
units)
          void set_scale(float scale = 1.f);

          // set OFFSET, the value that's subtracted from
the actual reading (tare weight)
          void set_offset(long offset = 0);

          // puts the chip into power down mode
          void power_down();

          // wakes up the chip after power down mode
          void power_up();
};

#endif /* HX711_h */
```

APPENDIX B. SPARKFUN HX711 LOAD CELL AMPLIFIER OPEN-SOURCE C++ SOURCE CODE FILE [51]

```
#include <Arduino.h>
#include <HX711.h>

HX711::HX711(byte dout, byte pd_sck, byte gain) {
     PD_SCK    = pd_sck;
     DOUT      = dout;

     pinMode(PD_SCK, OUTPUT);
     pinMode(DOUT, INPUT);

     set_gain(gain);
}

HX711::~HX711() {

}

bool HX711::is_ready() {
     return digitalRead(DOUT) == LOW;
}

void HX711::set_gain(byte gain) {
     switch (gain) {
          case 128:      // channel A, gain factor 128
               GAIN = 1;
               break;
          case 64:       // channel A, gain factor 64
               GAIN = 3;
               break;
          case 32:       // channel B, gain factor 32
               GAIN = 2;
               break;
     }

     digitalWrite(PD_SCK, LOW);
     read();
}

long HX711::read() {
     // wait for the chip to become ready
     while (!is_ready());
     byte data[3];

     // pulse the clock pin 24 times to read the data
     for (byte j = 3; j--;) {
          for (char i = 8; i--;) {
               digitalWrite(PD_SCK, HIGH);
               bitWrite(data[j], i, digitalRead(DOUT));
               digitalWrite(PD_SCK, LOW);
          }
     }

     // set the channel and the gain factor for the next
reading using the clock pin
     for (int i = 0; i < GAIN; i++) {
          digitalWrite(PD_SCK, HIGH);
          digitalWrite(PD_SCK, LOW);
     }

     data[2] ^= 0x80;

     return ((uint32_t) data[2] << 16) | ((uint32_t)
data[1] << 8) | (uint32_t) data[0];
}

long HX711::read_average(byte times) {
     long sum = 0;
     for (byte i = 0; i < times; i++) {
          sum += read();
     }
     return sum / times;
}

double HX711::get_value(byte times) {
     return read_average(times) - OFFSET;
}

float HX711::get_units(byte times) {
     return get_value(times) / SCALE;
}

void HX711::tare(byte times) {
     double sum = read_average(times);
     set_offset(sum);
}

void HX711::set_scale(float scale) {
     SCALE = scale;
}

void HX711::set_offset(long offset) {
     OFFSET = offset;
}

void HX711::power_down() {
     digitalWrite(PD_SCK, LOW);
     digitalWrite(PD_SCK, HIGH);
}

void HX711::power_up() {
     digitalWrite(PD_SCK, LOW);
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. ONBOARDLINK.INO

```
// Arduino On-board Software controlling a manipulator link (OnBoardLink.ino)
// Sends telemetry and receives commands via TCP/UDP.
// Developed by Dr. Josep Virgili-Llop and CPT Jerry Drew

// Through UDP the manipulator sends:
//   * Position in rad.
//   * Angular velocity in rad/s.
//   * Joint torque in Nm
//
// The manipulator accepts velocity commands in rad/s.
//

//--- CODE ---//

//Include libraries
#include <SPI.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#include "HX711.h" //for load cell amp to read torque

//Main loop time in ms
#define LOOP_TIME 100 // 10 Hz

//Wireless Parameters
int stat_w = WL_IDLE_STATUS;
#define ssid  "dlink_srl" //  Network SSID (name)
#define FSS_IP  "192.168.0.133" // IP of the FSS
#define localPort  4097 // Local port to listen on

//FSS ports (will be assigned depending on mac address)
#define FSS_PORT_A 25010
#define FSS_PORT_B 25020
#define FSS_PORT_C 25030
#define FSS_PORT_D 25040
byte mac[6];  //Holds Wi-Fi shied mac address
int FSS_port= FSS_PORT_A; //Holds FSS port

//Mac addresses of WIfi shields (in reverse!)
#define MAC_A 0xb1,0xc5,0x01,0x0e,0xc4,0x78
#define MAC_B 0x7a,0xfb,0x01,0x0e,0xc4,0x78
#define MAC_C 0x49,0xc5,0x01,0x0e,0xc4,0x78
#define MAC_D 0x47,0xc5,0x01,0x0e,0xc4,0x78
```

```
#define calibration_factor 1.0
#define DOUT  8 //Torque sensor DOUT pin #
#define CLK  9 //Torque sensor CLK pin #

//Motor Parameters
#define POS_Conversion 624339 // Conversion counts per radian
#define VEL_Conversion 624339 // Conversion counts/s to radians per second
#define CUR_Conversion 100    // Conversion from driver response to Amperes
#define COUNT_OFFSET 0        // Count offset to 0 degrees.

//Joint limits
#define POS_LIMIT 1.4835 // 85 degrees

//UDP packet buffer
byte packetBuffer[255]; // Buffer to hold incoming packet

//Receiver buffer for Serial to driver
String DriverString;

//Create Wi-Fi UDP class
WiFiUDP Udp;

// Definition of the data class that will hold the data to reveieve and send
typedef union {
 float floatPrec;
 byte binary[4];
} binaryFloat;

//Loop variables
binaryFloat Pos, Velocity, Torque, CS; //Data variables

//Torque sensor sariable
HX711 scale(DOUT, CLK);  //defining a variable of the class HX711 that is defined in
the X711.h library

//--- SETUP ---//
void setup() {

  //Initialize serial (debug)
  Serial.begin(9600);
  //Initialise serial port with Driver
  Serial2.begin(9600);
  Serial2.setTimeout(20); //
  Serial.println("Serial to Driver initialized");
  //Wait 5 s to allow Driver power-up
  delay(5000);

  //Initialize Driver
  DriverSetup();

  //Initialize Wifi
  WifiSetup();

  //Initialize UPD listening
  UDPSetup();

  //Torque Setup
  Serial.println("Initializing Torque Sensor");
  scale.set_scale(calibration_factor); //This value is obtained by using the
SparkFun_HX711_Calibration sketch
  scale.tare();  //Assuming there is no weight on the scale at start up, reset the scale to 0

  Serial.println("Initalization Complete");
}

//--- MAIN LOOP ---//
float command=0;
void loop() {

  //Initial loop time
  int t_init = millis();

  //Recieve velocity commands
  if (Udp.parsePacket()){
    //Read command
    command=VelCommand();
    //Within limits -> execute motion
    if (abs(Pos.floatPrec)<POS_LIMIT) {execute(command);}
    else if (Pos.floatPrec>POS_LIMIT & command<0) {execute(command);}
    else if (Pos.floatPrec<-POS_LIMIT & command>0) {execute(command);}
   }

   //Check if the joint limits are exceeded
   if (Pos.floatPrec>POS_LIMIT & Velocity.floatPrec>0) {execute(0);} //Exceeded limits
-> stop motion
   else if (Pos.floatPrec<-POS_LIMIT & Velocity.floatPrec<0) {execute(0);} //Exceeded
limits -> stop motion
```

```
  //Get Torque
  float  reading = scale.read_average(1);
  Torque.floatPrec = (reading-8388608)/487848.1*0.33;
  //Get joint position
  Pos.floatPrec=GetPos();
  //Get joint velocity
  Velocity.floatPrec=GetVelocity();

  //Compute checksum
  CS.floatPrec = 100-(Torque.floatPrec*100+Pos.floatPrec*100+Velocity.floatPrec*100);

  //Send data via UDP
  Udp.beginPacket(FSS_IP, FSS_port);
  //Udp.write(Time.binary,4);
  Udp.write(Pos.binary,4);
  Udp.write(Velocity.binary,4);
  Udp.write(Torque.binary,4);
  //Udp.write(Current.binary,4);
  Udp.write(CS.binary,4);
  int Udp_send = Udp.endPacket();

  //Print all the variables in a single line
  Serial.print("Angle: ");
  Serial.print(Pos.floatPrec*180/3.1415,4);
  Serial.print("deg, ");
  Serial.print("Velocity: ");
  Serial.print(Velocity.floatPrec*180/3.1415,4);
  Serial.print("deg/s, ");
  Serial.print("Torque: ");
  Serial.print(Torque.floatPrec,4);
  Serial.print("Nm, ");
  Serial.print("UDP Send: ");
  Serial.print(Udp_send);
  Serial.print(", Exec Time: ");
  Serial.print(millis()-t_init);
  Serial.println("ms");

  //Wait until total loop time has elapsed (loops executes at a constant rate)
  while ((LOOP_TIME+t_init)>millis()) {}
}


//--- FUNCTIONS ---//

void WifiSetup() {
```

```
// check for the presence of the shield:
delay(1000); //Wait for WiFi to boot up
if (WiFi.status() == WL_NO_SHIELD) {
  Serial.println("WiFi shield not present");
  // don't continue:
  while (true);
}

String fv = WiFi.firmwareVersion();
Serial.print("Wifi firmware version: ");
Serial.println(fv);
if ( fv != "1.1.0" )
  Serial.println("Please upgrade the firmware.");

// attempt to connect to Wifi network:
while ( stat_w != WL_CONNECTED) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  // Connect to network.
  stat_w = WiFi.begin(ssid);
  //Print result (for debugging)
  Serial.print("Wifi status: ");
  Serial.println(stat_w);

  // wait 2 seconds for connection:
  delay(2000);
}
Serial.println("Connected to wifi");

//Get your shield mac address
WiFi.macAddress(mac);

// Assign FSS port based on MAC address
byte mac_a[6] = {MAC_A};
byte mac_b[6] = {MAC_B};
byte mac_c[6] = {MAC_C};
byte mac_d[6] = {MAC_D};
if (memcmp(mac,mac_a,6)==0) {
  FSS_port = FSS_PORT_A;
}
else if (memcmp(mac,mac_b,6)==0) {
  FSS_port = FSS_PORT_B;
}
else if (memcmp(mac,mac_c,6)==0) {
    FSS_port = FSS_PORT_C;
  }
  else if (memcmp(mac,mac_d,6)==0) {
    FSS_port = FSS_PORT_D;
  }


  //Print wifi variables
  printWifiStatus();
}


//Display WIFI information and get Mac address
void printWifiStatus() {

  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print your WiFi shield's MAC address:
  Serial.print("MAC: ");
  Serial.print(mac[5],HEX);
  Serial.print(":");
  Serial.print(mac[4],HEX);
  Serial.print(":");
  Serial.print(mac[3],HEX);
  Serial.print(":");
  Serial.print(mac[2],HEX);
  Serial.print(":");
  Serial.print(mac[1],HEX);
  Serial.print(":");
  Serial.println(mac[0],HEX);

  // print the target FSS port
  Serial.print("FSS port: ");
  Serial.println(FSS_port);


  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

//Setup the Driver to Velocity Mode
void DriverSetup() {

  //Set all the variables required
  Serial.println("Setting up Current Mode.");

  //Set acceleration rate
  Serial.println("Setting acceleration rate.");
  Serial2.print("s r0x36 ");
  Serial2.print(50000); //Acceleration in counts/second^2
  Serial2.print("\n");
  DriverString = Serial2.readStringUntil('\r');
  Serial.println(DriverString);

  //Set decceleration rate
  Serial.println("Setting decceleration rate.");
  Serial2.print("s r0x37 ");
  Serial2.print(50000); //Decceleration in counts/second^2
  Serial2.print("\n");
  DriverString = Serial2.readStringUntil('\r');
  Serial.println(DriverString);

  //Set Velocity
  Serial.println("Setting Velocity");
  Serial2.print("s r0x2f ");
  Serial2.print(0); //Velocity in
  Serial2.print("\n");
  DriverString = Serial2.readStringUntil('\r');
  Serial.println(DriverString);

  //Enable Amplifier
  Serial.println("Enable Amplifier.");
  Serial2.print("s r0x24 11\n");
  DriverString = Serial2.readStringUntil('\r');
  Serial.println(DriverString);
}

//Get position joint data from the driver
float GetPos() {
  //Send command to driver asking for motor position.
  Serial2.print("g r0x32\n");
  //Read driver reply
  DriverString = Serial2.readStringUntil('\r');
  //Remove the first "v"
  DriverString = DriverString.substring(2);
  //Conver it to an angle in rad
  return (DriverString.toFloat()-COUNT_OFFSET)/POS_Conversion;
}


//Get joint velocity data from the driver
float GetVelocity() {

  //Send command to driver asking for motor velocity.
  Serial2.print("g r0x18\n");
  //Read driver reply
  DriverString = Serial2.readStringUntil('\r');
  //Remove the first "v"
  DriverString = DriverString.substring(1);
  //Conver it to an angular velocity in rad/s
  return DriverString.toFloat()/10/VEL_Conversion;
}

//Get joint velocity data from the driver
float GetCurrent() {

  //Send command to driver asking for motor current.
  Serial2.print("g r0x0c\n");
  //Read driver reply
  DriverString = Serial2.readStringUntil('\r');
  //Remove the first "v"
  DriverString = DriverString.substring(1);
  //Convert it to a current in Amps
  return DriverString.toFloat()/CUR_Conversion;

}

//Send velocity command to driver
void execute(float Velocity) {

  //Set Velocity
  Serial2.print("s r0x2f ");
  Serial2.print(Velocity*VEL_Conversion*10,0); //Velocity in 0.1 counts per second
  Serial2.print("\n");
  //Read driver response
  DriverString = Serial2.readStringUntil('\r');
}

//Initialize UDP connection
void UDPSetup(){

  Serial.println("Starting connection to server...");
  // if you get a connection, report back via serial:
  if (Udp.begin(localPort) == 1) Serial.println("started");
  else Serial.println("failed");
}

//Read velocity command from UDP
float VelCommand() {

  //Read velocity command from UDP stream
   binaryFloat Velocity;
   Udp.read(Velocity.binary,4);
   return Velocity.floatPrec;
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. MARSMAN_DRIVER_LOAD.CCX

```
14
1
95,0,Host  Config  State,2:-1:0:1792:-1:-1:150:-1:-1:-1:-1:-
1:-1:-1:-1:-1:-1:-1:-1:-1
87,0,Amp Family,3
40,0,Motor Type,48
89,0,Amp Max Voltage,910
84,0,Amp ISense,1050
83,0,Amp Continuous Current,300
82,0,Amp Peak Current,900
88,0,Amp Peak Current Time,1000
0,0,Current Cp,55
1,0,Current Ci,19
2,0,Programmed Current Command,0
19,0,Analog Input Scale,425
1a,0,Analog Input Offset,0
21,0,User Peak Current Limit,425
22,0,User Continuous Current Limit,225
23,0,User Peak Current Time Limit,1000
24,0,Desired State,11
26,0,Analog Input Deadband,0
27,0,Velocity Vp,9984
28,0,Velocity Vi,10000
2f,0,Programmed Velocity Command,0
31,0,Velocity Gains Scalor,8
36,0,Velocity Loop Acceleration Limit,500
37,0,Velocity Loop Deceleration Limit,500
39,0,Velocity Fast Stop Ramp,131072
3a,0,Velocity Loop Velocity Limit,131072000
3f,0,Velocity Tracking Time,100
3e,0,Velocity Tracking Window,393216
41,0,Motor Manufacturer,Harmonic Drive Systems
42,0,Motor Model Number,FHA-8C-30-12S17bE
43,0,Motor Units,0
44,0,Motor Inertia,29000
46,0,Motor Brake,1
48,0,Motor Torque Constant,3060
49,0,Motor Resistance,108
4a,0,Motor Inductance,44
4b,0,Motor Peak Torque,13000
4c,0,Motor Continuous Torque,6900
4d,0,Motor Velocity Limit,131072000
4e,0,Motor Wiring,1
```

```
53,0,Brake/Stop Delay Time,0
54,0,Motor Brake Delay Time,0
55,0,Motor Brake Activation Velocity,0
56,0,Motor Back Emf Constant,320
6a,0,Commanded Current Ramp,10000
70,0,Output 1 Config,100:4000:0
78,0,Input 1 Config,17
79,0,Input 2 Config,0
7a,0,Input 3 Config,0
7b,0,Input 4 Config,0
7c,0,Input 5 Config,0
7d,0,Input 6 Config,0
80,0,Amp Model Number,DEP-090-09
86,0,Servo Period,4
8a,0,Voltage Sense,2262
92,0,Amp Name,Current
98,0,Function Generator Config,2
99,0,Function Generator Frequency,0
9a,0,Function Generator Amplitude,0
9b,0,Function Generator Duty Cycle,1000
a5,0,Input Configuration Register,0
a7,0,Fault Mask,1567
a8,0,Digital Command Config,0
a9,0,Digital Command Scaling,1
ad,0,Hardware Type,897
ae,0,Current Loop Offset,0
30,0,Position Pp,1000
33,0,Position Vff,16384
34,0,Position Aff,0
45,0,Motor Pole Pairs,5
4f,0,Motor Hall Offset,-60
50,0,Motor Hall Type,0
52,0,Motor Hall Wiring,0
5f,0,Velocity  Loop  Output  Filter,8448:200:0:775:1550:775:-
12774:32763:5813
60,0,Motor Encoder Type,14
61,0,Motor Encoder Units,0
62,0,Motor Encoder Counts,131072
63,0,Motor Encoder Resolution,1000
64,0,Motor Electrical Distance,100000
65,0,Motor Encoder Direction,1
6c,0,Position Capture Control Register,5
71,0,Output 2 Config,100:44007f:0
8e,0,Amp Ref Scale,11220
af,0,Amp Options Register,2
b1,0,Increment Rate,0
```

```
b2,0,Commutation Mode,6
ba,0,Position Following Error Limit,131072
bb,0,Position Following Warning Limit,65536
bc,0,Position Tracking Window,32768
bd,0,Position Tracking Window Time,10
c8,0,Trajectory Profile Mode,0
ca,0,Trajectory Position Command,1
cb,0,Trajectory Max Velocity,32768000
cc,0,Trajectory Max Accel,1310720
cd,0,Trajectory Max Decel,1310720
cf,0,Trajectory Abort Decel,6553600
f0,0,Input 1 Debounce Time,0
f1,0,Input 2 Debounce Time,0
f2,0,Input 3 Debounce Time,0
f3,0,Input 4 Debounce Time,0
f4,0,Input 5 Debounce Time,0
f5,0,Input 6 Debounce Time,0
114,0,Velocity Vi Drain,0
10c,0,Network Heart Beat Time,0
10d,0,CANopen Node Guarding Time,0
10e,0,CANopen Node Guarding Life Time Factor,0
109,0,Camming Master Velocity,0
105,0,Camming Configuration,4352
106,0,Camming Forward Delay,0
107,0,Camming Reverse Delay,0
104,0,Phase Init Config,0
be,0,Software Limit Deceleration,655360
e4,0,Phase Init Current,425
e5,0,Phase Init Time,400
58,0,Gear Ratio,65537
b6,0,PWM Deadband,1000
ea,0,Detent Correction Gain,0
57,0,Micro Steps Per Rev.,0
59,0,Hall Velocity Shift,0
5a,0,Multi Mode Port Configuration,1
5b,0,Position Encoder Resolution,4000
5c,0,Position Encoder Direction,0
5d,0,Position Encoder Type,0
67,0,Analog Encoder Shift,0
6b,0,Velocity         Loop         Command         Filter,-
7936:200:0:775:1550:775:-12774:32763:5813
6f,0,PWM Mode,0
72,0,Output 3 Config,0:0:0
73,0,Output 4 Config,0:0:0
7e,0,Input 7 Config,0
7f,0,Input 8 Config,0
```

```
b3,0,Analog Encoder Scale,6667
b8,0,Positive Software Limit,10000
b9,0,Negative Software Limit,1000
bf,0,Home Current Delay Time,250
c1,0,Node ID configuration,1024
c2,0,Home Configuration,512
c3,0,Home Velocity Fast,3276800
c4,0,Home Velocity Slow,655360
c5,0,Home Accel/Decel,655360
c6,0,Home Offset,0
c7,0,Home Current,113
ce,0,Trajectory Max Jerk,26214400
d0,0,Input 9 Config,0
d1,0,Input 10 Config,0
d2,0,Input 11 Config,0
d3,0,Input 12 Config,0
d8,0,Regen Resistance,0
d9,0,Regen Continuous Power,0
da,0,Regen Peak Power,0
db,0,Regen Peak Power Time,0
e1,0,Regen Resistor Model Number,None
e3,0,Position Loop Gains Multiplier,100
e8,0,uStep Holding Current,0
e9,0,uStep Run To Hold Time,0
f6,0,Input 7 Debounce Time,0
f7,0,Input 8 Debounce Time,0
f8,0,Input 9 Debounce Time,0
f9,0,Input 10 Debounce Time,0
fa,0,Input 11 Debounce Time,0
fb,0,Input 12 Debounce Time,0
103,0,Network address Input Map,0
11a,0,Amp Scaling Config,0
121,0,Network Options,0
123,0,Motor Position Wrap Value,0
124,0,Load Position Wrap Value,0
125,0,MACRO Amplifier's Encoder Capture Config,0
127,0,Gain Scheduling Config,0
12a,0,Motor Encoder Options,301994001
12b,0,Position Encoder Options,1
12d,0,Analog     Input     Filter,-7936:200:0:775:1550:775:-
12774:32763:5813
10f,0,Registration Offset For Pulse and Direction,0
13c,0,Minimum PWM Pulse Width PWM Position Mode,1000
13d,0,Maximum PWM Pulse Width PWM Position Mode,2000
100,0,CANopen limit mask,25364352
94c,0,basic host cfg,33
```

APPENDIX E. VICON_CALIBRATE.M

```
% Vicon_Calibrate.m captures the VICON data and saves it to an m-file
% Developed by multiple users within the Spacecraft Robotics Laboratory
% Adapted by Dr. Josep Virgili-Llop and CPT Jerry Drew

% The data is saved in the following format:
% [X [m], Y [m], Z [m], EulerX [rad], EulerY [rad], EulerZ [rad], Time
[s]]
%
% Needs the Vicon SDK added in the path for it to work
(http://www.vicon.com/products/software/datastream-sdk).

%--- Clean and Clear ---%
clc
clear all
close all

%--- Object Name ---%

%Vicon objects names
obj_name = {'Prime2','EndEffector'}; %Prime2 is FSS



%--- Load Vicon Matlab SDK ---%
fprintf( 'Loading Vicon SDK...\n' );
addpath('C:\Program Files\Vicon\DataStream SDK\Win64\MATLAB')
Client.LoadViconDataStreamSDK();
fprintf( 'done\n' );

%--- Create Client and connect to localhost ---%

%Create Client
ViconClient = Client();

%Connect to localhost (where Tracker is streaming the data)
HostName = 'localhost:801';
fprintf( 'Connecting to %s ...', HostName );
while ~ViconClient.IsConnected().Connected
  % Direct connection
  ViconClient.Connect( HostName );
end

%--- Configure Vicon Client ---%

% Enable some different data types
ViconClient.EnableSegmentData();
ViconClient.EnableMarkerData();
ViconClient.EnableUnlabeledMarkerData();
ViconClient.EnableDeviceData();
```

```
%Check if data types are enabled
fprintf( 'Segment Data Enabled: %d\n',
ViconClient.IsSegmentDataEnabled().Enabled );
fprintf( 'Marker Data Enabled: %d\n',
ViconClient.IsMarkerDataEnabled().Enabled );


% Set the streaming mode
ViconClient.SetStreamMode( StreamMode.ClientPull );
ViconClient.SetAxisMapping( Direction.Forward, ...
                         Direction.Left,    ...
                         Direction.Up );    % Z-up

%--- Capture Data ---%

%A dialog to stop the loop
MessageBox = msgbox( 'Stop Capture', 'Vicon data capture' );
%A dialog to capture data
CaptureBox = msgbox( 'Capture', 'Vicon data capture' );

%Initialize data structure
EE_meas=[]; %state vector of end effector [3x1]
q0_meas=[]; %state vector of base [3x1]

% Loop until the message box is dismissed
while ishandle( MessageBox )

    %Update message box
    drawnow;

    % Get frame
    while ViconClient.GetFrame().Result.Value ~= Result.Success

    end

    for i=1:length(obj_name)
        %Retrieve object values
        Output_GetSegmentGlobalTranslation =
ViconClient.GetSegmentGlobalTranslation( char(obj_name(i)),
char(obj_name(i)) );
        Output_GetSegmentGlobalRotationEulerXYZ =
ViconClient.GetSegmentGlobalRotationEulerXYZ( char(obj_name(i)),
char(obj_name(i)) );

        %Save data
        data(i,1:3) = [Output_GetSegmentGlobalTranslation.Translation(
1 )/1e3, ... %x
            Output_GetSegmentGlobalTranslation.Translation( 2 )/1e3,
...            %y
            Output_GetSegmentGlobalRotationEulerXYZ.Rotation( 3 )];
%theta z

        %Display latest data
        if i==1; clc; end
        fprintf('Object: %s\n',char(obj_name(i)));
        fprintf('Position x,y,z: [%7.3f,%7.3f]
[m]\n',data(i,1),data(i,2));
        fprintf('Euler Angles  : [%7.2f] [deg]\n',data(i,3)*180/pi);
    end

    %Compute Manipulator data
    q0 = data(1,:);
    EE = data(2,:);



    if not(ishandle( CaptureBox ))
        q0_meas(end+1,1:3)=q0;
        EE_meas(end+1,1:3)=EE;
        CaptureBox = msgbox( 'Capture', 'Vicon data capture' );
    end

end

%--- Disconnect and unload ---%

% Disconnect and dispose
fprintf( 'Disconnect...\n' );
ViconClient.Disconnect();
% Unload the SDK
fprintf( 'Unload SDK...\n' );
Client.UnloadViconDataStreamSDK();

%--- Ask to save data ---%

%Generate filename
filename=sprintf('Vicon_%s_%s.mat','MARSMAN_CAL',datestr(now,30));
uisave({'q0_meas','EE_meas'},filename);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. KINEMATICS_SERIAL.M

```
% Kinematics_Serial.m computes the kineamtics of a serial manipulator.
% Developed by Dr. Josep Virgili-Llop

function
[RJ,RL,r,l,e,t0,tm,Bij,Bi0,P0,pm,TEE]=Kinematics_Serial(R0,r0,qm,q0dot,
qmdot,data)
% Input ->
%   R0 -> Rotation matrix from the base-spacecraft to the inertial
frame.
%   r0 -> Position of the base-spacecraft to the inertial frame.
%   qm -> Manipulator joint varibles.
%   q0dot -> Base-spacecraft velocities [angular velocity in body,
linear
%   velocity in inertial].
%   qmdot -> Manipulator joint rates.
%   data -> Manipulator data.
%       data.n -> Manipulator number of joints and links.
%       data.base -> Base-spacecraft data
%           data.base.T_L0_J1 -> Homogeneous transformation of the
first
%           joint w.r.t. the base-spacecraft.
%       data.man -> Manipulator data.
%           data.man(i).DH -> DH parameters of the ith joint.
%           data.man(i).type -> Type of joint. type==0 for revolute,
%           otherwise prismatic.
%           data.man(i).b -> Vector from the ith link to the following
%           joint i+1.
%
% Output ->
%   RJ -> Joint 3x3 rotation matrices.
%   RL -> Links 3x3 rotation matrices.
%   r -> Links positions.
%   l -> Joints positions.
%   e -> Joints rotations axis.
%   t0 -> Base-spacecraft twist vector
%   tm -> Manipulator twist vector.
%   Bij -> Twist-propagation matrix (for manipulator i>0 and j>0).
%   Bi0 -> Twist-propagation matrix (for i>0 and j=0).
%   P0 -> Base-spacecraft twist-propagation vector.
%   pm -> Manipulator twist-propagation vector.
%   TEE -> End-Effector Homogeneous transformation matrix.

%=== LICENSE ===%

%=== CODE ===%

%--- Number of links and Joints ---%
n=data.n;

%--- Homogeneous transformation matrices ---%
```

```
%Pre-allocate homogeneous transformations matrices
TJ=zeros(4,4,n+1);
TL=zeros(4,4,n);
%Base-spacecraft
T0 = [R0,r0;zeros(1,3),1];
%First Joint
TJ(1:4,1:4,1) =T0*data.base.T_L0_J1;
%Forward recursive for rest of joints and links
for i=1:n
    %Compute Rotation matrix and translation vector from DH parameters
    [R,s] = DH_Rs(data.man(i).DH,qm(i),data.man(i).type);
    %Compute joint homogeneous transformation matrix
    TJ(1:4,1:4,i+1)=TJ(1:4,1:4,i)*[R,s;zeros(1,3),1];
    %Compute link homogeneous transformation matrix
    TL(1:4,1:4,i)=TJ(1:4,1:4,i+1)*[eye(3),-data.man(i).b; zeros(1,3),
1]; %homog trans matrix of last link
end
%End-Effector
TEE = TJ(1:4,1:4,n+1); %3x3 rotation matrix plus 3x1 translation vector
describing position of EE

%--- Rotation matrices, translation, position and other geometry
vectors ---%
%Pre-allocate rotation matrices, translation and position vectors
RJ=zeros(3,3,n);
RL=zeros(3,3,n);
r=zeros(3,n);
l=zeros(3,n);
%Pre-allocate axis
e=zeros(3,n);
%Pre-allocate other gemotery vectors
g=zeros(3,n);
%Format Rotation matrices, link positions, joint axis and other
geometry
%vectors
for i=1:n
    RJ(1:3,1:3,i)=TJ(1:3,1:3,i);
    RL(1:3,1:3,i)=TL(1:3,1:3,i);
    r(1:3,i)=TL(1:3,4,i);
    e(1:3,i)=RJ(1:3,3,i);
    l(1:3,i)=TJ(1:3,4,i);
    g(1:3,i)=r(1:3,i)-l(1:3,i);
end

%--- Twist-propagtaion matrix ---%
%Pre-allocate Bij
Bij=zeros(6,6,n,n);
%Compute Bij
for j=1:n
    for i=1:n
        Bij(1:6,1:6,i,j)=[eye(3), zeros(3,3); SkewSym(r(1:3,j)-
r(1:3,i)), eye(3)];
    end
end
%Pre-allocate Bi0
```

```
Bi0=zeros(6,6,n);
%Compute Bi0
for i=1:n
    Bi0(1:6,1:6,i)=[eye(3), zeros(3,3); SkewSym(r0-r(1:3,i)), eye(3)];
end

%--- Twist-Propagation vector ---%
%Pre-allocate
pm=zeros(6,n);
%Base-spacecraft
P0=[R0,zeros(3,3); zeros(3,3), eye(3)];
%Fordward recursion to obtain the Twist-Propagation vector
for i=1:n
    if data.man(i).type==0
        %Revolute joint
        pm(1:6,i)=[e(1:3,i);cross(e(1:3,i),g(1:3,i))];
    else
        %Prismatic joint
        pm(1:6,i).p=[zeros(3,1);e(1:3,i)];
    end
end

%--- Generalized twist vector ---%
%Pre-Allocate
tm=zeros(6,n);
%Base-spacecraft
t0=P0*q0dot;
%First link
tm(1:6,1)=Bi0(1:6,1:6,1)*t0+pm(1:6,1)*qmdot(1);
%Fordward recursion to obtain the twist vector
for i=2:n
    tm(1:6,i)=Bij(1:6,1:6,i,i-1)*tm(1:6,i-1)+pm(1:6,i)*qmdot(i);
end

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. DH_RS.M

```
% This function computes the rotation matrix R and the translation
vector s
% between two joints given their Denavit-Hartenber (DH) parameters.
% Developed by Dr. Josep Virgili-Llop

function [R,s] = DH_Rs(DH,qm,type)
%
% Inputs:
%   DH -> Denavit-Hartenberg parameters.
%       DH.d -> Distance between joint origins along the joint z-axis.
%       DH.theta -> Rotation between x-axis along the joint z-axis.
%       DH.alpha -> Rotation between z-axis.
%       DH.a -> Distance between the commaon normal between the z-axis.
%   qm -> Joint variable.
%   type -> type==0 for revolute joint or type==1 for prismatic joints.
%
% Outputs:
%   R -> Rotation 3x3 matrix.
%   s -> Translation 3x1 vector.

%=== LICENSE ===%

%=== CODE ===%

%Assign the d and theta variable depending on joint type
if type==0
    %Revolute joint
    theta=qm;
    d=DH.d;
else
    %Prismatic joint
    theta=Dh.theta;
    d=qm;
end



%Rotation matrix
R = [
    cos(theta), -sin(theta)*cos(DH.alpha), sin(theta)*sin(DH.alpha);
    sin(theta), cos(theta)*cos(DH.alpha),  -cos(theta)*sin(DH.alpha);
    0,               sin(DH.alpha),                 cos(DH.alpha)
    ];

%Translation vector
s = [DH.a*cos(theta), DH.a*sin(theta), d]';

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. SKEWSYM.M

```
% SkewSym.m computes the skewsymmetric matrix of a vector
% Developed by Dr. Josep Virgili-Llop

function [x_skew] = SkewSym(x)

x_skew=[0 -x(3) x(2) ; x(3) 0 -x(1) ; -x(2) x(1) 0 ];

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. MARSMAN_KINCAL.M

```
%MARSMAN_KinCal.m calibrates the kinematic model based on actually
measured end effector
%positions

%Developed by Dr. Josep Virgili-Llop and CPT Jerry Drew

% Inputs:
%   q  -> measured angular positions of each joint
%   delta_a -> induced variation in DH.a
%   delta_theta -> induced variation in DH.theta
%   delta_q -> induced variation in joint position
%   delta_x -> induced variation in x direction displacement
%   DH -> Denavit-Hartenberg parameters.
%       DH.d -> Distance between joint origins along the joint z-axis.
%       DH.theta -> Rotation between x-axis along the joint z-axis.
%       DH.alpha -> Rotation between z-axis.
%       DH.a -> Distance between the commaon normal between the z-axis.
%   EE_Eul_Ang -> angular component of vector from origin of inertial
frame to end
%                 effector, scalar (rad)
%   Eps -> convergence error tolerance for calibration
%   I -> inertia of manipulator links
%   qm -> Joint variable.
%   mass -> mass of manipulator links (kg)
%   PHI -> kinematic calibration matrix
%   r_EE_x -> x component of vector from origin of inertial frame to
end
%             effector, scalar (m)
%   r_EE_y -> y component of vector from origin of inertial frame to
end
%             effector, scalar (m)
%   RBJ1 -> rotation matrix about z of joint 1 with respect to the base
%   theta_base_J1 -> angular displacement of base with respect to joint
1
%                    (inertial frame)
%   type -> type==0 for revolute joint or type==1 for prismatic joints.
%   x_base_J1 -> displacement of base with respect to joint 1 in x
%                direction (inertial frame)
%   y_base_J1 -> displacement of base with respect to joint 1 in y
%                direction (inertial frame)
%
% Outputs:
%   Delta_Zeta_Direct -> matrix containing computed variations between
%                        measured and actual DH parameters

clc
clear
close all
```

```
%Load Data
load ('Kinematic_Data/Calibration_data_March3.mat')
load ('Kinematic_Data/Vicon_MARSMAN_CAL_20160303T111525.mat')

%---  Prepare data ---%
q_meas=q;
q_meas(3:4,:) = []; %Remove unwanted measurements
q=[]; %Delete this variable as it is going to be used later on.


%--- Define manipulator data ---%

%Number of joints/links
data.n=4;

%First joint
data.man(1).type=0;
data.man(1).DH.d = 0;
data.man(1).DH.alpha = 0;
data.man(1).DH.a = .39; %length from joint(i) to joint(i+1), meters
data.man(1).b = [data.man(1).DH.a/2;0;0];
data.man(1).mass=2.88;
data.man(1).I=eye(3)*0.04;
data.man(1).q0=0;

%Second joint
data.man(2)=data.man(1);

%Third joint
data.man(3)=data.man(1);

%Fourth joint
data.man(4)=data.man(1);


%Base to first link data (initial guess)
theta_base_J1 = 0;
x_base_J1 = 0.2;
y_base_J1 = 0;

%First joint location with respect to base
RBJ1=[cos(theta_base_J1)   -sin(theta_base_J1)  0; %Rotation matrix
about z
    sin(theta_base_J1)  cos(theta_base_J1)  0;
    0                    0                   1];
data.base.T_L0_J1=[RBJ1,[x_base_J1;y_base_J1;0];zeros(1,3),1];

%Base-spacecraft inertia matrix
data.base.mass=12;
data.base.I=eye(3)*0.22;

%--- LSQ Parameters ---%
Eps = 0.01; % convergence error tolerance
```

```
Delta_Zeta_Direct = 2*Eps;
%Variations
delta_d = 1e-3; %Distance variation
delta_theta = deg2rad(.01); %Angle variation

%Initial guess on joint offsets
for j=1:size(q_meas,1)
    for i=1:data.n
        q(j,i)=q_meas(j,i)+data.man(i).q0;
    end
end

%--- Iterative Batch LSQ ---%
while max(abs(Delta_Zeta_Direct)) > Eps;

    %Nominal Values
    [rEE_x, rEE_y, rEE_theta] = x_EE(q0_meas, q, data); % kinematic
model estimate of end effector state

    %--- Derivatives for DH.a for all links except the last one ---%
    %It is assumed that all links have the same length (except the last
one)
    %Copy Manipulator data
    data_var=data;
    %Change DH.a
    for i = 1:data.n-1
        data_var.man(i).DH.a = data.man(i).DH.a + delta_d;
    end
    %Values with varied parameters
    [rEE_x_delta, rEE_y_delta, rEE_theta_delta] = x_EE(q0_meas, q,
data_var); % compute position of EE under variation
    %Partial derivatives matrix
    PHI(:,1) = [((rEE_x_delta - rEE_x)/delta_d)'; ((rEE_y_delta -
rEE_y)/delta_d)'; (rEE_theta_delta(:)-rEE_theta(:))/delta_d];

    %--- Derivatives for DH.a for the last link ---%
    %Copy Manipulator data
    data_var=data;
    %Change DH.a
    data_var.man(end).DH.a = data.man(end).DH.a + delta_d;
    %Values with varied parameters
    [rEE_x_delta, rEE_y_delta, rEE_theta_delta] = x_EE(q0_meas, q,
data_var); % compute position of EE under variation
    %Partial derivatives matrix
    PHI(:,2) = [((rEE_x_delta - rEE_x)/delta_d)'; ((rEE_y_delta -
rEE_y)/delta_d)'; (rEE_theta_delta(:)-rEE_theta(:))/delta_d];

    %--- Derivatives with respect to Base to First Joint parameters ---
%

    %Angle
    data_var=data;
    theta_base_J1_var = theta_base_J1 + delta_theta;
    RBJ1=[cos(theta_base_J1_var)   -sin(theta_base_J1_var)  0; %Resets
RBJ1 to baseline
        sin(theta_base_J1_var)  cos(theta_base_J1_var)  0;
        0                     0                   1];

    data_var.base.T_L0_J1(1:3,1:3)=RBJ1;
    %Values with varied parameters
    [rEE_x_delta, rEE_y_delta, rEE_theta_delta] = x_EE(q0_meas, q,
data_var); % compute position of EE under variation
    %Partial derivatives matrix
    PHI(:,3) = [((rEE_x_delta - rEE_x)/delta_theta)'; ((rEE_y_delta -
rEE_y)/delta_theta)'; (rEE_theta_delta(:)-rEE_theta(:))/delta_theta];

    %X position of first joint
    data_var=data;
    data_var.base.T_L0_J1(1,4)=x_base_J1 + delta_d;
    %Values with varied parameters
    [rEE_x_delta, rEE_y_delta, rEE_theta_delta] = x_EE(q0_meas, q,
data_var); % compute position of EE under variation
    %Partial derivatives matrix
    PHI(:,4) = [((rEE_x_delta - rEE_x)/delta_d)'; ((rEE_y_delta -
rEE_y)/delta_d)'; (rEE_theta_delta(:)-rEE_theta(:))/delta_d];

    %X position of first joint
    data_var=data;
    data_var.base.T_L0_J1(2,4)=y_base_J1 + delta_d;
    %Values with varied parameters
    [rEE_x_delta, rEE_y_delta, rEE_theta_delta] = x_EE(q0_meas, q,
data_var); % compute position of EE under variation
    %Partial derivatives matrix
    PHI(:,5) = [((rEE_x_delta - rEE_x)/delta_d)'; ((rEE_y_delta -
rEE_y)/delta_d)'; (rEE_theta_delta(:)-rEE_theta(:))/delta_d];

    %Joint angle Offsets
    for i = 2:data.n
        data_var=data;
        data_var.man(i).q0=data_var.man(i).q0+delta_theta;
        q_var=q;
        for j=1:size(q_meas,1)
            q_var(j,i)=q_meas(j,i)+data_var.man(i).q0;
        end
        %Values with varied parameters
        [rEE_x_delta, rEE_y_delta, rEE_theta_delta] = x_EE(q0_meas,
q_var, data_var); % compute position of EE under variation
        %Partial derivatives matrix
        PHI(:,i+5-1) = [((rEE_x_delta - rEE_x)/delta_theta)';
((rEE_y_delta - rEE_y)/delta_theta)'; (rEE_theta_delta(:)-
rEE_theta(:))/delta_theta];
    end

    %---- Create Delta_x ---%
    x_n = [rEE_x' ; rEE_y'; rEE_theta'];
    x_m = [EE_meas(:,1); EE_meas(:,2); EE_meas(:,3)];
    Delta_x = x_m - x_n;
    %--- Solve LSQ ---%
    Delta_Zeta_Direct = inv(PHI'*PHI)*PHI'*Delta_x;


    %--- Update parameters ---%
    %Update links lengths
    for i = 1:data.n-1
        data.man(i).DH.a = data.man(i).DH.a + Delta_Zeta_Direct(1);
    end
    data.man(end).DH.a = data.man(end).DH.a + Delta_Zeta_Direct(2);
    %Update position of first joint w.r.t base
    theta_base_J1 = theta_base_J1+ Delta_Zeta_Direct(3);
    x_base_J1 = x_base_J1 + Delta_Zeta_Direct(4);
    y_base_J1 = y_base_J1 + Delta_Zeta_Direct(5);
    RBJ1=[cos(theta_base_J1)   -sin(theta_base_J1)  0;
        sin(theta_base_J1)  cos(theta_base_J1)  0;
        0                    0                  1];
    data.base.T_L0_J1=[RBJ1,[x_base_J1;y_base_J1;0];zeros(1,3),1];
    %Joint offsets
    for i = 2:data.n
        data.man(i).q0=data.man(i).q0+Delta_Zeta_Direct(5+i-1);
    end
    for j=1:size(q_meas,1)
        for i=1:data.n
            q(j,i)=q_meas(j,i)+data.man(i).q0;
        end
    end


end

%--- Print Results ---%
for i = 1:data.n
    fprintf('Link %i DH a: %f m\n',i,data.man(i).DH.a);
end
for i = 1:data.n
    fprintf('Link %i Joint Offset: %f
deg\n',i,rad2deg(data.man(i).q0));
end
fprintf('L0 to J1 parameters [x,y,theta]: %f m, %f m, %f
deg.\n',x_base_J1,y_base_J1,rad2deg(theta_base_J1));

%--- Print Residuals ---%
fprintf('\n');
%Compute kineamtics with last iteration
[rEE_x, rEE_y, rEE_theta] = x_EE(q0_meas, q, data);
%Compute residuals (mean and std)
x_res=(rEE_x'-EE_meas(:,1));
y_res=(rEE_y'-EE_meas(:,2));
pos_res = sqrt(x_res.^2+y_res.^2);
fprintf('position residuals [mean,std]: %f, %f
m.\n',mean(pos_res),std(pos_res));
%Compute residuals (mean and std)
```

```
theta_res=(rEE_theta'-EE_meas(:,3));
fprintf('theta residuals [mean,std]: %f, %f
deg.\n',rad2deg(mean(theta_res)),rad2deg(std(theta_res)));

%---Update b vector ---%
for i = 1:data.n
    data.man(i).b = [data.man(i).DH.a/2;0;0];
end

%--- Save manipulator Data (ready for dynamic calibration) ---%
save('KinCal_ManData','data');
```

APPENDIX J. X_EE.M

```
%Comupte the state of the end-effector according to the kinematic model
and
%the joint states (base and manipulator joints).

% Developed by Dr. Josep Virgili-Llop and CPT Jerry Drew

function [rEE_x, rEE_y, rEE_theta] = x_EE(q0_meas, q, data)

%Iterate throught the different states
for i = 1:length(q)

    %Base rotation matrix
    R0=[cos(q0_meas(i,3))   -sin(q0_meas(i,3))  0;
        sin(q0_meas(i,3))  cos(q0_meas(i,3))  0;
        0                   0                  1];

    %Base position
    r0=[q0_meas(i,1); %translation in x
        q0_meas(i,2); %translation in y
        0];           %translation in z

    %Joint variables
    qm = q(i,:); %manipulator joint states [m x n] where m is the
number of measurements taken and n is the number of links

    %--- Compute Kinematics, Dynamics, ID, and FD ---%

    %Kinematics

[~,~,~,~,~,~,~,~,~,~,~,TEE]=Kinematics_Serial(R0,r0,qm,zeros(6,1),zeros
(data.n,1),data);
    rEE_theta(i) = dcm2angle(TEE(1:3,1:3)'); %orientations of EE wrt
inertial frame
    rEE_x(1,i) = TEE(1,4);
    rEE_y(1,i) = TEE(2,4);

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K. MARSMAN_DYNCAL.M

```
% Performs the dynamic calibration based upon the dynamic model,
% measurement data, and the results of the kinematic calibration.

% Developed by Dr. Josep Virgili-Llop and CPT Jerry Drew

% Inputs:
%   delta_d -> distance variation of b vector
%   delta_m -> mass variation (kg)
%   delta_I -> inertia variation (kg*m^2)
%   Delta_x -> difference between measured and nominal base angular
position
%   Eps -> convergence error tolerance for calibration
%   qm1 -> joint positions before maneuver
%   qm2 -> joint positions after maneuver
%   q01 -> state (x,y,theta) of base before maneuver
%   q02 -> state (x,y,theta) of base after maneuver
%   q0_px -> nominal (predicted) x value of base from dynamic model
%   q0_py -> nominal (predicted) y value of base from dynamic model
%   q0_ptheta -> nominal (predicted) theta value of base from dynamic
model
%   xEE1 -> end effector state before maneuver
%   xEE2 -> end effector state after maneuver
%   x_m -> measured theta value of base
%   x_n -> nominal (predicted) theta value of base from dynamic model
%         (q0_ptheta')
%   I -> inertia of manipulator links
%   PHI -> dynamic calibration matrix

% Outputs:
%   Delta_Zeta_Direct -> matrix containing parameter variations with
%                        respect to the nominal values (Siciliano, p.
89);
%                        the solution to the iterative least-squares
%                        problem

clc
clear
close all

%Load Data
load ('KinCal_ManData.mat') %loads results from kinematic calibration

for i = 1:15

    load (sprintf('DynCal_Data/qmat%i.mat',i)) % loads telemetry joint
measurements
    load (sprintf('DynCal_Data/Man_%i.mat',i)) % loads VICON base and
EE measurements

    %Initial guess on joint offsets
    for j=1:size(q,1)
        for k=1:data.n
            q(j,k)=q(j,k)+data.man(k).q0; % accounts for offsets
determined by kinematic calibration
        end
    end

    maneuver(i).q01 = q0_meas (1,1:3); % state (x,y,theta) of base at
beginning of maneuver
    maneuver(i).q02 = q0_meas (2,1:3); % state (x,y,theta) of base at
end of maneuver
    maneuver(i).qm1 = q (1,:); % joint positions before maneuver
    maneuver(i).qm2 = q (2,:); % joint positions after maneuver
    maneuver(i).xEE1 = EE_meas (1,1:3); % end effector state before
maneuver
    maneuver(i).xEE2 = EE_meas (2,1:3); % end effector state after
maneuver

end

maneuver([6,7,8,15])=[];
maneuver([4,8,10,11])=[];

%--- LSQ Parameters ---%
Eps = 0.01; % convergence error tolerance
Delta_Zeta_Direct = 2*Eps;
%Variations
delta_d = 1e-3; % Distance variation of b vector
delta_m = 1e-3; % Mass variation (kg)
delta_I = 1e-5; % Inertia variation (kg*m^2)



%--- Iterative Batch LSQ ---%
while max(abs(Delta_Zeta_Direct)) > Eps;

    %Nominal Values
    [q0_px, q0_py, q0_ptheta] = q0_maneuver(maneuver, data); % nominal
(predicted) values from dynamic model

    %--- Derivatives for DH.I for all links ---%
    %Copy Manipulator data
    data_var=data;
    %Change DH.I
    for i = 1:data.n
        data_var.man(i).I(3,3) = data.man(i).I(3,3) + delta_I; %
changing z component of Inertia matrix (in local frame)
    end
    %Values with varied parameters
    [q0_px_delta, q0_py_delta, q0_ptheta_delta] = q0_maneuver(maneuver,
data_var);
    PHI(:,1) = [((q0_ptheta_delta - q0_ptheta)/delta_I)'];
```

```
%     %--- Derivatives for DH.I for the last link ---%
%     %Copy Manipulator data
%     data_var=data;
%
%     data_var.man(end).I(3,3) = data.man(end).I(3,3) + delta_I;
%
%     %Values with varied parameters
%     [q0_px_delta, q0_py_delta, q0_ptheta_delta] =
q0_maneuver(maneuver, data_var);
%     PHI(:,2) = [((q0_ptheta_delta - q0_ptheta)/delta_I)'];


    %---- Create Delta_x ---%
    x_n = q0_ptheta';
    for i=1:length(maneuver)
        x_mx(i,1)=maneuver(i).q02(1);
        x_my(i,1)=maneuver(i).q02(2);
        x_mtheta(i,1)=maneuver(i).q02(3);
    end
    x_m = x_mtheta;
    Delta_x = wrapToPi(x_m - x_n);

    %--- Solve LSQ ---%
    Delta_Zeta_Direct = inv(PHI'*PHI)*PHI'*Delta_x;


    %--- Update parameters ---%
    %Update links Inertia
    for i = 1:data.n
        data.man(i).I(3,3) = data.man(i).I(3,3) +
Delta_Zeta_Direct(1)/10;
    end

end

%--- Print Results ---%
for i = 1:data.n
    fprintf('Link %i I: %f kg m^2\n',i,data.man(i).I(3,3));
end
for i = 1:data.n
    fprintf('Link %i b: %f m\n',i,data.man(i).b(1));
end

%--- Print Residuals ---%
fprintf('\n');
%Compute kineamtics with last iteration
[q0_px, q0_py, q0_ptheta] = q0_maneuver(maneuver, data);
%Compute residuals (mean and std)
x_n = q0_ptheta';
q0_theta_res= rad2deg(wrapToPi(x_m - x_n));
fprintf('position residuals [mean,std]: %f, %f
deg.\n',mean(q0_theta_res),std(q0_theta_res));
```

```
%--- Save manipulator Data (ready for dynamic calibration) ---%
save('DynCal_ManData','data');
```

APPENDIX L. I_I.M

```
% Converts the inertias in local frame to inertia in the inertial
frame.
% Developed by Dr. Josep Virgili-Llop

function [I0,Im]=I_I(R0,RL,data)
% Input ->
%   R0 -> Rotation matrix from the base-spacecraft to the inertial
frame.
%   RL -> Links 3x3 rotation matrices.
%       data.n -> Manipulator number of joints and links.
%       data.man -> Manipulator data.
%       data.man(i).I -> Link inertia.
% Output ->
%   I0 -> Base-spacecraft inertia in inertial frame.
%   Im -> Manipulator inertia in inertial frame.

%=== LICENSE ===%

%=== CODE ===%

%Base-spacecraft inertia
I0 = R0*data.base.I;
%Pre-allocate inertias
Im=zeros(3,3,data.n);
%Inertias of the links
for i=1:(data.n)
    Im(1:3,1:3,i)=RL(1:3,1:3,i)*data.man(i).I*RL(1:3,1:3,i)';
end

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX M. MCB_SERIAL.M

```
% Computes the Mass Composite Body matrix of a Serial Manipulator.
% Developed by Dr. Josep Virgili-Llop

function [M0_tilde,Mm_tilde]=MCB_Serial(I0,Im,Bij,Bi0,data)
% Input ->
%   I0 -> Base-spacecraft inertia in inertial frame.
%   Im -> Manipulator inertia in inertial frame.
%   Bij -> Twist-propagation matrix (for manipulator i>0 and j>0).
%   Bi0 -> Twist-propagation matrix (for i>0 and j=0).
%       data.base -> Base-spacecraft data
%           data.base.mass -> Base-spacecraft mass.
%       data.man -> Manipulator data.
%           data.man.n -> Manipulator number of joints and links.
%           data.man(i).mass -> Link mass.
% Output ->
%   M0_tilde -> Base-spacecraft mass matrix of composite body.
%   Mm_tilde -> Manipulator mass matrix of composite body.

%=== LICENSE ===%

%=== CODE ===%
%Number of links and Joints
n=data.n;
%Pre-allocate
Mm_tilde=zeros(6,6,n);
%Initialize M tilde
Mm_tilde(1:6,1:6,n)=[Im(1:3,1:3,n),zeros(3,3);zeros(3,3),data.man(n).ma
ss*eye(3)];
%Backwards recursion
for i=n-1:-1:1

Mm_tilde(1:6,1:6,i)=[Im(1:3,1:3,i),zeros(3,3);zeros(3,3),data.man(i).ma
ss*eye(3)]+Bij(1:6,1:6,i+1,i)'*Mm_tilde(1:6,1:6,i+1)*Bij(1:6,1:6,i+1,i)
;
end
%Base-spacecraft M tilde
M0_tilde=[I0,zeros(3,3);zeros(3,3),data.base.mass*eye(3)]+Bi0(1:6,1:6,1
)'*Mm_tilde(1:6,1:6,1)*Bi0(1:6,1:6,1);

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX N. GIM_SERIAL.M

```
% Computes the Generalized Inertia Matrix of a Serial Manipulator.
% Developed by Dr. Josep Virgili-Llop

function [H0, H0m, Hm] =
GIM_Serial(M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,data)
%
% Input ->
%   M0_tilde -> Base-spacecraft mass matrix of composite body.
%   Mm_tilde -> Manipulator mass matrix of composite body.
%   Bij -> Twist-propagation matrix (for manipulator i>0 and j>0).
%   Bi0 -> Twist-propagation matrix (for i>0 and j=0).
%   P0 -> Base-spacecraft twist-propagation vector.
%   pm -> Manipulator twist-propagation vector.
%   data -> Manipulator data.
%       data.n -> Manipulator number of joints and links.
%
% Output ->
%   H0 -> Base-spacecraft inertia matrix.
%   H0m -> Base-spacecraft - manipulator coupling inertia matrix.
%   Hm -> Manipulator inertia matrix.

%=== LICENSE ===%



%=== CODE ===%

%--- Number of links and Joints ---%
n=data.n;

%--- H Martix ---%
%Base-spacecraft Inertia matrix
H0 = P0'*M0_tilde*P0;
%Pre-allocate Hm
Hm=zeros(n,n);
%Manipulator Inertia matrix Hm
for j=1:n
    for i=j:n

Hm(i,j)=pm(1:6,i)'*Mm_tilde(1:6,1:6,i)*Bij(1:6,1:6,i,j)*pm(1:6,j);
        Hm(j,i)=Hm(i,j);
    end
end
%Pre-allocate H0m
H0m=zeros(6,n);
%Coupling Inertia matrix
for i=1:n
    H0m(1:6,i)=(pm(1:6,i)'*Mm_tilde(1:6,1:6,i)*Bi0(1:6,1:6,i)*P0)';
end

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX O. Q_DOT_FUN.M

```
% Computes time derivatives of the base-spacecraft state vector (q_dot)
% Developed by Dr. Josep Virgili-Llop

function [q_dot] = q_dot_fun(t,q,qmdot,maneuver,data)
%
% Inputs:
%   Bij -> Twist-propagation matrix (for manipulator i>0 and j>0).
%   Bi0 -> Twist-propagation matrix (for i>0 and j=0).
%   H0 -> Base-spacecraft inertia matrix.
%   H0m -> Base-spacecraft - manipulator coupling inertia matrix.
%   Hm -> Manipulator inertia matrix.
%   I0 -> Base-spacecraft inertia in inertial frame.
%   Im -> Manipulator inertia in inertial frame.
%   M0_tilde -> Base-spacecraft mass matrix of composite body.
%   Mm_tilde -> Manipulator mass matrix of composite body.
%   P0 -> Base-spacecraft twist-propagation vector.
%   pm -> Manipulator twist-propagation vector.
%   R0 -> Rotation matrix from the base-spacecraft to the inertial
frame.
%   r0 -> Position of the base-spacecraft to the inertial frame.
%
% Outputs:
%  q_dot -> derivative of the state vector of the spacecraft base and
%           manipulator joints, of the form [q0_dot;qm_dot]



%--- Derivative of qm ---%
q_dot(4:3+data.n)=qmdot;

%--- Derivative of q0 ---%
%Base rotation matrix
R0=[cos(maneuver.q01(3))   -sin(maneuver.q01(3))  0;
    sin(maneuver.q01(3))  cos(maneuver.q01(3))  0;
    0                   0                  1];

%Base position
r0=[maneuver.q01(1); %translation in x
    maneuver.q01(2); %translation in y
    0];           %translation in z

%Kinematics
[~,RL,~,~,~,~,~,Bij,Bi0,P0,pm,~]=Kinematics_Serial(R0,r0,q(4:end),zeros
(6,1),zeros(data.n,1),data);
%Inertias
[I0,Im]=I_I(R0,RL,data);
%Mass Composite Body matrix
[M0_tilde,Mm_tilde]=MCB_Serial(I0,Im,Bij,Bi0,data);
%Generalized Inertia matrix
[H0, H0m, ~] = GIM_Serial(M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,data);
```

```
q0_dot= -inv(H0) * H0m * qmdot';
q_dot(1:3) = q0_dot([4,5,3]);

q_dot=q_dot(:);


end
```

APPENDIX P. Q0_MANEUVER.M

```
% q0_maneuver.m predicts the coordinates of the base's position in the
inertial frame by
% integrating the joint velocities.
%
% Developed by Dr. Josep Virgili-Llop and CPT Jerry Drew
%
% Inputs:
%   maneuver -> structure containing position data of base, joints, and
%               end effector before and after maneuver
%   data -> structure containing number of links, DH parameters, mass,
%           inertias, types of joints, and initial state vector of the
%           system
%
% Outputs:
%    q0_px -> nominal position data of base and joints (x coordinates)
%    q0_py -> nominal position data of base and joints (y coordinates)
%    q0_theta -> nominal position data of base and joints (theta
coordinates)

function [q0_px, q0_py, q0_ptheta] = q0_maneuver(maneuver,data)

Delta_t=5.0;

for i = 1:length(maneuver)

    qmdot = (maneuver(i).qm2-maneuver(i).qm1)/Delta_t; %assume
denominator is t = 1

    ode_deriv=@(t,q)q_dot_fun(t,q,qmdot,maneuver(i),data); % q=[q0,qm]
    options = odeset('RelTol',1e-12,'AbsTol',1e-12);
    [t,q0]=ode45(ode_deriv,[0:1e-
3:Delta_t],[maneuver(i).q01,maneuver(i).qm1],options);


    q0_px(i)=q0(end,1);
    q0_py(i)=q0(end,2);
    q0_ptheta(i)=q0(end,3);

end

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] M. Oda, K. Kibe, and F. Yamagata, "ETS-VII, space robot in-orbit experiment satellite," in *Proc. IEEE International Conference on Robotics and Automation (ICRA),* Minneapolis, 1996, vol. 1, pp. 739–744.

[2] A. Ogilvie, J. Allport, M. Hannah, and J. Lymer, "Autonomous satellite servicing using the Orbital Express demonstration manipulator system," in *Proc. 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS'08)*, Los Angeles, CA, 2008, pp. 25–29.

[3] M. J. Mataric, *The Robotics Primer,* Cambridge, MA: MIT Press, 2007.

[4] K. Shamaei, Y. Che, A. Murali, S. Sen, S. Patil, K. Goldberg, and A. M. Okamura, "A paced shared-control teleoperate architecture for supervised automation of multilateral surgical tasks," in *Proc. 2015 IEEE Intelligent Robots and Systems (IROS),*Hamburg, Germany, 2015, pp. 1434–1439.

[5] L. G. Torres, A. Kuntz, H. B. Gilbert, P. J. Swaney, R. J. Hendrick, R. J. Webster, and R. Alterovitz, "A motion planning approach to automatic obstacle avoidance during concentric rube robot teleoperation," in 2015 *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, 2015, pp. 2361–2367.

[6] S. Vozar, S. Leonard, P. Kazanzides, and L. L. Whitcomb, "Experimental evaluation of force control for virtual-fixture-assisted teleoperation for on-orbit manipulation of satellite thermal blanket insulation," in 2015 *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, 2015, pp. 4424–4431

[7] Rethink Robotics. "Baxter with Intera 3." [Online]. Available: http://www.rethinkrobotics.com/baxter. Accessed Apr. 14, 2016.

[8] J. Long, *The Great Courses: Robotics*, Chantilly, VA: The Teaching Company, 2015.

[9] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, and P. Abbeel, K. Goldberg, "Learning by observation for surgical subtasks: multilateral cutting of 3D viscoelastic and 2D orthotropic tissue phantoms," in *2015 International Conference on Robotics and Automation (ICRA),* Seattle, WA 2015, pp. 1202–1209.

[10] Canadian Space Agency. "Canadarm." [Online]. Available: http://www.asc-csa.gc.ca/eng/canadarm/. Accessed Apr. 13, 2016.

[11] Canadian Space Agency. “Historic first moves,” [Online]. Available: http://www.asc-csa.gc.ca/eng/canadarm/beginning.asp. Accessed April 14, 2016.

[12] K. McBryan and D. Akin, “Mission overview of the dynamic manipulator flight experiment (DYMAFLEX): a nanosatellite test bed to study coupled dynamics between a robotic arm and an equivalently-sized small host vehicle in the space environment,” in *63rd International Astronautical Congress,* Turin, Italy, 2012.

[13] K. Yoshida, “Experimental study of the dynamics and control of a space robot with experimental free-floating robot satellite (EFFORTS) simulators,” in *Advanced Robotics,* Japan, VSP and Robotics Society of Japan, 1995, pp. 583–602.

[14] M. Wilde, M. Romano, and A. Fleischner, “Historical review of spacecraft formation flight, docking, and space robot manipulation simulators,” in preparation for publication.

[15] K. Machida, T. Yoshitsugu, and T. Iwata, “Maneuvering and manipulation of flying space telerobotics system,” in *1992 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Raleigh, NC, 1992, vol.1, pp. 3–10.

[16] Deutsches Zentrum für Luft- und Raumfahrt (DLR), “ROTEX (1993),” [Online]. Available: http://www.dlr.de/rmc/rm/en/desktopdefault.aspx/tabid-3827/5969_read-8744. Accessed Apr. 14, 2016.

[17] Japan Aerospace Exploration Agency, “About Engineering Test Satellite VII ‘KIKU-7 (ETS-VII),” [Online]. Available: http://global.jaxa.jp/projects/sat/ets7/index.html. Accessed Apr. 14, 2016.

[18] I. Kawano, M. T. Suzuki, H. Koyama, and M. Kunugi, “Approach trajectory design for autonomous rendezvous of ETS-VII,” *Journal of the Japan Society for Aeronautical and Space Sciences*, 2001, vol. 49, no. 575, pp. 432–437.

[19] N. D’Amore and D. Akin, “Space manipulator control for the DYMAFLEX flight experiment,” Air Force Office of Scientific Research, Arlington, VA, Tech. Rep. AFRL-OSR-VA-TR-2013-0468, Sep. 2013.

[20] D. Alvarez, “Design, integration, and test of a modular spacecraft-based robotic manipulator link,” M.S. thesis, Dept. Astro. Eng., Naval Postgraduate School, Monterey, CA, 2014.

[21] P. Boning, M. Ono, T. Nohara, and S. Dubowsky, “An experimental study on the control of space robot teams assembling large flexible structures,” in *Proc. 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS’12),* Montreal, Canada.

[22] X. Wei, S. Fuchun, and L. Huaping, “Design and development of a ground experiment system with free-flying robot,” in *Proc. 6th Conference on Industrial Electronics and Applications (ICIEA),* Beijing, China, 2011, pp. 2101–2016.

[23] J. Virgili-Llop, J. Drew, and M. Romano, “Spacecraft robotics toolkit: an open-source simulator for spacecraft robotic arm dynamic modeling and control,” presented at 6th International Conference on Astrodynamics Tools and Techniques (ICATT), Darmstadt, Germany, 2016.

[24] J. Virgili-Llop and J. Drew. “SPAcecraft robotics toolkit (SPART).” [Online]. Available: https://github.com/NPS-SRL. Accessed Feb. 23, 2016.

[25] Harmonic Drive LLC. “HD LLC ASCII interface programmer’s guide.” [Online]. Available: www.harmonicdrive.net/_hd/.../ASCII-PROGRAMMERS-GUIDE.pdf. Accessed Mar. 18, 2016.

[26] Futek Inc. “Futek model TFF400.” [Online]. Available: http://www.futek.com/files/pdf/Product%20Drawings/tff400.pdf. Accessed Jan. 4, 2016].

[27] Inspired Energy. “Battery specification, document number and revision: DSNH2054HD31 [Online]. Available: http://www.inspired-energy.com/standard_products/NH2054/NH2054.htm. Accessed Apr. 4, 2016.

[28] Traco Power. “DC/DC converters, TEP 75WI, 75W.” [Online]. Available: http://www.tracopower.com/products/tep75wi.pdf. Accessed Apr. 4, 2016.

[29] Copley Controls. “Software documents, CME – setup & indexing.” [Online]. Available: http://www.copleycontrols.com/Motion/Downloads/software.html. Accessed Apr. 4, 2016.

[30] Futek Inc. “Futek wiring codes,” [Online]. Available: http://www.futek.com/code_wire.aspx. Accessed Jan. 19, 2016.

[31] Arduino Inc. “Arduino Due” [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardDue. Accessed Apr. 4, 2016.

[32] LinkSprite. “RS232 shield V2 for Arduino.” [Online]. Available: http://store.linksprite.com/rs232-shield-v2-for-arduino/. Accessed Apr. 4, 2016.

[33] Sparkfun Inc. “SparkFun load cell amplifier–HX711.” [Online]. Available: https://www.sparkfun.com/products/13230. Accessed Jan. 12, 2016.

[34] N. Seidle. “Sparkfun_HX711_Load_Cell_v10.” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/SparkFun_HX711_Load_Cell_v10.pdf. Accessed Jan. 21, 2016.

[35] Arduino Inc. “Arduino Wi-Fi shield.” [Online]. Available: https://www.arduino.cc/en/Main/ArduinoWiFiShield. Accessed Jan. 12, 2016.

[36] Adafruit. “5V 1.5A linear voltage regulator -7805 TO-220.” Available: https://www.adafruit.com/products/2164. Accessed Apr. 22, 2016.

[37] Harmonic Drive LLC. “DEP.” [Online]. Available: http://www.harmonicdrive.net/products/servo-drives/dc-bus/dep. Accessed Apr. 21, 2016.

[38] All About Circuits. “Wiring color codes,” [Online]. Available: http://www.allaboutcircuits.com/textbook/reference/chpt-2/wiring-color-codes. Accessed Apr. 21, 2016.

[39] Harmonic Drive LLC. “FHA mini series servo actuators.” [Online]. Available: http://www.harmonicdrive.net/_hd/content/catalogs/pdf/fha-c-miniture-ac100v-200v.pdf. Accessed Apr. 21, 2016.

[40] Sparkfun. “RS-232 vs. TTL serial communication,” [Online]. Available: https://www.sparkfun.com/tutorials/215. Accessed Apr. 22, 2016.

[41] Harmonic Drive LLC. “Digital servo drive for brushless or brush motors, DEP Series,” [Online]. Available: http://www.harmonicdrive.net/_hd/content/documents/dep-series.pdf. Accessed Apr. 22, 2016.

[42] R. Zappula, J. Virgili-Llop, H. Park, and M. Romano, “Experiments on autonomous spacecraft rendezvous and docking using an artificial potential field approach,” presented at AAS/AIAA Space Flight Mechanics Meeting, Napa, CA, 2016.

[43] V. Gite. “What is the difference between UDP and TCP internet protocols?” [Online]. Available: http://www.cyberciti.biz/faq/key-differences-between-tcp-and-udp-protocols. Accessed Apr. 25, 2016.

[44] J. Virgili-Llop, J. Drew, and M. Romano, “Design and parameter identification by laboratory experiments of a prototype modular robotic arm for orbiting spacecraft applications,” presented at 6th International Conference on Astrodynamics Tools and Techniques (ICATT), Darmstadt, Germany, 2016.

[45] VICON. “Datastream SDK.” [Online]. Available: http://www.vicon.com/products/software/datastream-sdk. Accessed Apr. 6, 2016.

[46] J. Diebel. “Representing attitude: Euler angles, unit quaternions, and rotation vectors.” [Online], Available: http://www.swarthmore.edu/NatSci/mzucker1/e27/diebel2006attitude.pdf. Accessed Apr. 25, 2016.

[47] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modeling, Planning, and Control,* London, England: Springer, 2010.

[48] J. Virgili-Llop, M. Wilde, M. Romano, "A detailed tutorial on the modeling of orbiting spacecraft with an on-board robotic manipulator," accepted for presentation at AIAA SPACE and Astronautics Forum and Exposition, Long Beach, CA, 2016.

[49] J. Virgili-Llop, J. Drew, and M. Romano, "Autonomous capture of an object by a spacecraft with a modular robotic am: analysis, simulation, and experiments," in preparation for publication, 2016.

[50] Yale University. "Yale OpenHand project." [Online]. Available: http://www.eng.yale.edu/grablab/openhand. Accessed Mar. 23, 2016.

[51] Github Inc. "An Arduino library to interface the Avia Semiconductor HX711 24-bit analog-to-digital converter (ADC) for weight scales," [Online]. Available: https://github.com/bogde/HX711. Accessed Jan. 21, 2016.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California